1 Read this—it will help

Contents

1.1 Getting started with Stata

To get started with Stata, you should

1. Read the *Getting Started* (GSM, GSU, or GSW) manual for your operating system.
2. Then turn to the other manuals; see **[U] 1.2 The Stata Documentation**.

1.2 The Stata Documentation

The *User's Guide* is divided into three sections: *Stata basics*, *Elements of Stata*, and *Advice*. The table of contents lists the chapters within each of these sections. Click on the chapter titles to see the detailed contents of each chapter.

The *Guide* is full of a lot of useful information about Stata; we recommend that you read it. If you only have time, however, to read one or two chapters, then read **[U] 11 Language syntax** and **[U] 12 Data**.

The other manuals are the *Reference* manuals. The Stata *Reference* manuals are each arranged like an encyclopedia—alphabetically. Look at the *Base Reference Manual*. Look under the name of a command. If you do not find the command, look in the subject index in [I] *Stata Index*. A few commands are so closely related that they are documented together, such as `ranksum` and `median`, which are both documented in [R] **ranksum**.

Not all the entries in the *Base Reference Manual* are Stata commands; some contain technical information, such as [R] **Maximize**, which details Stata's iterative maximization process, or [R] **Error messages**, which provides information on error messages and return codes.

Like an encyclopedia, the *Reference* manuals are not designed to be read from cover to cover. When you want to know what a command does, complete with all the details, qualifications, and pitfalls, or when a command produces an unexpected result, read its description. Each entry is written at the level of the command. The descriptions assume that you have little knowledge of Stata's features when they are explaining simple commands, such as those for using and saving data. For more complicated commands, they assume that you have a firm grasp of Stata's other features.

If a Stata command is not in the *Base Reference Manual*, you can find it in one of the other *Reference* manuals. The titles of the manuals indicate the types of commands that they contain. The *Programming Reference Manual*, however, contains commands not only for programming Stata but also for manipulating matrices (not to be confused with the matrix programming language described in the *Mata Reference Manual*).

The complete list of Stata Documentation is as follows:

[GSM]	*Getting Started with Stata for Mac*
[GSU]	*Getting Started with Stata for Unix*
[GSW]	*Getting Started with Stata for Windows*
[U]	*Stata User's Guide*
[R]	*Stata Base Reference Manual*
[ADAPT]	*Stata Adaptive Designs: Group Sequential Trials Reference Manual*
[BAYES]	*Stata Bayesian Analysis Reference Manual*
[BMA]	*Stata Bayesian Model Averaging Reference Manual*
[CAUSAL]	*Stata Causal Inference and Treatment-Effects Estimation Reference Manual*
[CM]	*Stata Choice Models Reference Manual*
[D]	*Stata Data Management Reference Manual*
[DSGE]	*Stata Dynamic Stochastic General Equilibrium Models Reference Manual*
[ERM]	*Stata Extended Regression Models Reference Manual*
[FMM]	*Stata Finite Mixture Models Reference Manual*
[FN]	*Stata Functions Reference Manual*
[G]	*Stata Graphics Reference Manual*
[H2OML]	*Machine Learning in Stata Using H2O: Ensemble Decision Trees Reference Manual*
[IRT]	*Stata Item Response Theory Reference Manual*
[LASSO]	*Stata Lasso Reference Manual*
[XT]	*Stata Longitudinal-Data/Panel-Data Reference Manual*
[META]	*Stata Meta-Analysis Reference Manual*
[ME]	*Stata Multilevel Mixed-Effects Reference Manual*
[MI]	*Stata Multiple-Imputation Reference Manual*
[MV]	*Stata Multivariate Statistics Reference Manual*
[PSS]	*Stata Power, Precision, and Sample-Size Reference Manual*
[P]	*Stata Programming Reference Manual*
[RPT]	*Stata Reporting Reference Manual*
[SP]	*Stata Spatial Autoregressive Models Reference Manual*
[SEM]	*Stata Structural Equation Modeling Reference Manual*
[SVY]	*Stata Survey Data Reference Manual*
[ST]	*Stata Survival Analysis Reference Manual*
[TABLES]	*Stata Customizable Tables and Collected Results Reference Manual*
[TS]	*Stata Time-Series Reference Manual*
[I]	*Stata Index*
[M]	*Mata Reference Manual*

In addition, installation instructions may be found in the *Installation Guide*.

1.2.1 PDF manuals

Every copy of Stata comes with Stata's complete PDF documentation.

The PDF documentation may be accessed from within Stata by selecting **Help** > **PDF documentation**. Even more convenient, every help file in Stata links to the equivalent manual entry. If you are reading **help regress**, simply click on `(View complete PDF manual entry)` below the title of the help file to go directly to the [R] **regress** manual entry.

We provide some tips for viewing Stata's PDF documentation at https://www.stata.com/support/faqs/resources/pdf-documentation-tips/.

1.2.1.1 Video example

PDF documentation in Stata

1.2.2 Example datasets

Various examples in this manual use what is referred to as the automobile dataset, `auto.dta`. We have created a dataset on the prices, mileages, weights, and other characteristics of 74 automobiles and have saved it in a file called `auto.dta`. (These data originally came from the April 1979 issue of *Consumer Reports* and from the United States Government EPA statistics on fuel consumption; they were compiled and published by Chambers et al. [1983].)

In our examples, you will often see us type

```
. use https://www.stata-press.com/data/r19/auto
```

We include the `auto.dta` file with Stata. If you want to use it from your own computer rather than via the internet, you can type

```
. sysuse auto
```

See [D] **sysuse**.

You can also access `auto.dta` by selecting **File > Example datasets...**, clicking on *Example datasets installed with Stata*, and clicking on `use` beside the `auto.dta` filename.

There are many other example datasets that ship with Stata or are available over the web. Here is a partial list of the example datasets included with Stata:

`auto.dta`	1978 automobile data
`bplong.dta`	Fictional blood-pressure data, long form
`bpwide.dta`	Fictional blood-pressure data, wide form
`cancer.dta`	Patient survival in drug trial
`census.dta`	1980 Census data by state
`citytemp.dta`	US city temperature data
`educ99gdp.dta`	Education and gross domestic product
`gnp96.dta`	US gross national product, 1967–2002
`lifeexp.dta`	1998 life expectancy
`network1.dta`	Fictional network diagram data
`nlsw88.dta`	1988 US National Longitudinal Survey of Young Women (NLSW), extract
`pop2000.dta`	2000 US Census population, extract
`sandstone.dta`	Subsea elevation of Lamont sandstone in an area of Ohio
`sp500.dta`	S&P 500 historic data
`surface.dta`	NOAA sea surface temperature
`tsline1.dta`	Simulated time-series data
`uslifeexp.dta`	US life expectancy, 1900–1999
`voter.dta`	1992 US presidential voter data

All of these datasets may be used or described from the **Example datasets...** menu listing.

Even more example datasets, including most of the datasets used in the reference manuals, are available at the Stata Press website (https://www.stata-press.com/data/). You can download the datasets with your browser, or you can use them directly from the Stata command line:

```
. use https://www.stata-press.com/data/r19/nlswork
```

An alternative to the `use` command for these example datasets is `webuse`. For example, typing

```
. webuse nlswork
```

is equivalent to the above `use` command. For more information, see [D] **webuse**.

1.2.2.1 Video example

Example datasets included with Stata

1.2.3 Cross-referencing

The *Getting Started* manual, the *User's Guide*, and the *Reference* manuals cross-reference each other.

[R] **regress**
[D] **reshape**
[XT] **xtreg**

The first is a reference to the `regress` entry in the *Base Reference Manual*, the second is a reference to the `reshape` entry in the *Data Management Reference Manual*, and the third is a reference to the `xtreg` entry in the *Longitudinal-Data/Panel-Data Reference Manual*.

[GSW] **B Advanced Stata usage**
[GSM] **B Advanced Stata usage**
[GSU] **B Advanced Stata usage**

are instructions to see the appropriate section of the *Getting Started with Stata for Windows*, *Getting Started with Stata for Mac*, or *Getting Started with Stata for Unix* manual.

1.2.4 The index

The *Stata Index* contains a combined index for all the manuals.

To find information and commands quickly, you can use Stata's `search` command; see [R] **search**. At the Stata command prompt, type `search geometric mean`. `search` searches Stata's keyword database and the internet to find more commands and extensions for Stata written by Stata users.

1.2.5 The subject table of contents

A subject table of contents for the *User's Guide* and all the *Reference* manuals is located in the *Stata Index*. This subject table of contents may also be accessed by clicking on **Contents** in the PDF bookmarks.

1.2.6 Typography

We mix the ordinary typeface that you are reading now with a typewriter-style typeface `that looks like this`. When something is printed in the typewriter-style typeface, it means that something is a command or an option—it is something that Stata understands and something that you might actually type into your computer. Differences in typeface are important. If a sentence reads, "You could list the result …", it is just an English sentence—you *could* list the result, but the sentence provides no clue as to how you might actually do that. On the other hand, if the sentence reads, "You could `list` the result …", it is telling you much more—you could list the result, and you could do that by using the `list` command.

We will occasionally lapse into periods of inordinate cuteness and write, "We `describe`d the data and then `list`ed the data." You get the idea. `describe` and `list` are Stata commands. We purposely began the previous sentence with a lowercase letter. Because `describe` is a Stata command, it must be typed in lowercase letters. The ordinary rules of capitalization are temporarily suspended in favor of preciseness.

We also mix in words printed in italic type, such as "To perform the rank-sum test, type `ranksum` *varname*`,` `by(`*groupvar*`)`". Italicized words are not supposed to be typed; instead, you are to substitute another word for them.

We would also like users to note our rule for punctuation of quotes. We follow a rule that is often used in mathematics books and British literature. The punctuation mark at the end of the quote is included in the quote only if it is a part of the quote. For instance, the pleased Stata user said she thought that Stata was a "very powerful program". Another user simply said, "I love Stata."

In this manual, however, there is little dialogue, and we follow this rule to precisely clarify what you are to type, as in, type "`cd c:`". The period is outside the quotation mark because you should not type the period. If we had wanted you to type the period, we would have included two periods at the end of the sentence: one inside the quotation and one outside, as in, type "the orthogonal polynomial operator, `p.`".

We have tried not to violate the other rules of English. If you find such violations, they were unintentional and resulted from our own ignorance or carelessness. We would appreciate hearing about them.

We have heard from Nicholas J. Cox of the Department of Geography at Durham University, UK, and express our appreciation. His efforts have gone far beyond dropping us a note, and there is no way with words that we can fully express our gratitude.

1.2.7 Vignette

If you look, for example, at the entry [R] **brier**, you will see a brief biographical vignette of Glenn Wilson Brier (1913–1998), who did pioneering work on the measures described in that entry. A few such vignettes were added without fanfare in the Stata 8 manuals, just for interest, and many more were added in Stata 9, and even more have been added in each subsequent release. A vignette could often appropriately go in several entries. For example, George E. P. Box deserves to be mentioned in entries other than [TS] **arima**, such as [R] **boxcox**. However, to save space, each vignette is given once only, and an index of all vignettes is given in the *Stata Index*.

Most of the vignettes were written by Nicholas J. Cox, Durham University, and were compiled using a wide range of reference books, articles in the literature, internet sources, and information from individuals. Especially useful were the dictionaries of Upton and Cook (2014) and Everitt and Skrondal (2010) and the compilations of statistical biographies edited by Heyde and Seneta (2001) and Johnson and Kotz (1997). Of these, only the first provides information on people living at the time of publication.

1.3 What's new

There are a lot of new features in Stata 19.

For a thorough overview of the most important new features, visit

https://www.stata.com/new-in-stata/

For a brief overview of all the new features that were added with the release of Stata 19, in Stata type

```
. help whatsnew18to19
```

Stata is continually being updated. For a list of new features that have been added since the release of Stata 19, in Stata type

```
. help whatsnew19
```

1.4 References

Chambers, J. M., W. S. Cleveland, B. Kleiner, and P. A. Tukey. 1983. *Graphical Methods for Data Analysis*. Belmont, CA: Wadsworth.

Everitt, B. S., and A. Skrondal. 2010. *The Cambridge Dictionary of Statistics*. 4th ed. Cambridge: Cambridge University Press.

Gould, W. W. 2014. Putting the Stata Manuals on your iPad. *The Stata Blog: Not Elsewhere Classified*. https://blog.stata.com/2014/10/28/putting-the-stata-manuals-on-your-ipad/.

Heyde, C. C., and E. Seneta, eds. 2001. *Statisticians of the Centuries*. New York: Springer.

Johnson, N. L., and S. Kotz, eds. 1997. *Leading Personalities in Statistical Sciences: From the Seventeenth Century to the Present*. New York: Wiley.

Pinzon, E., ed. 2015. *Thirty Years with Stata: A Retrospective*. College Station, TX: Stata Press.

Upton, G. J. G., and I. T. Cook. 2014. *A Dictionary of Statistics*. 3rd ed. Oxford: Oxford University Press.

2 A brief description of Stata

Stata is a statistical package for managing, analyzing, and graphing data.

Stata is available for a variety of platforms. Stata may be used either as a point-and-click application or as a command-driven package.

Stata's GUI provides an easy interface for those new to Stata and for experienced Stata users who wish to execute a command that they seldom use.

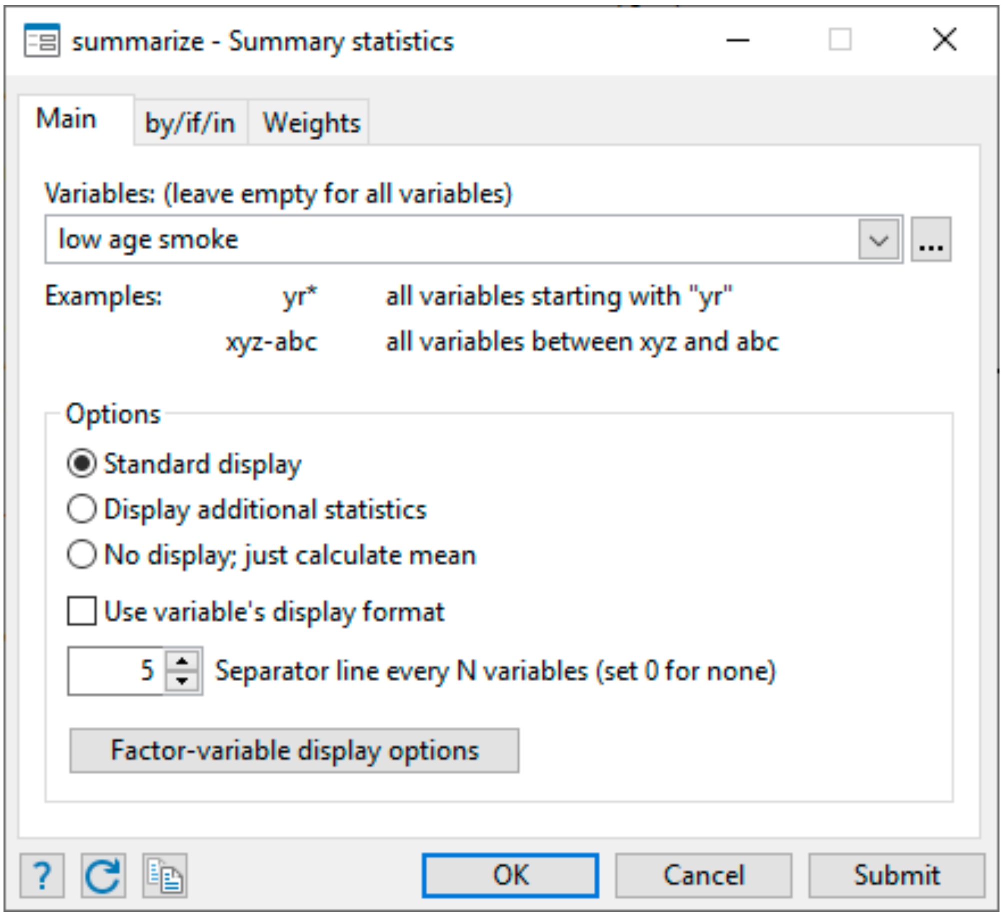
The command language provides a fast way to communicate with Stata and to communicate more complex ideas.

Here is an extract of a Stata session using the GUI:

(Throughout the Stata manuals, we will refer to various datasets. These datasets are all available from https://www.stata-press.com/data/r19/. For easy access to them within Stata, type `webuse` *dataset_name*, or select **File > Example datasets...** and click on *Stata 19 manual datasets*.)

```
. webuse lbw
(Hosmer & Lemeshow data)
```

We select **Data > Describe data > Summary statistics** and choose to summarize variables `low`, `age`, and `smoke`, whose names we obtained from the Variables window. We click on **OK**.


summarize - Summary statistics

Main | by/if/in | Weights

Variables: (leave empty for all variables)

low age smoke

Examples: yr* all variables starting with "yr"
xyz-abc all variables between xyz and abc

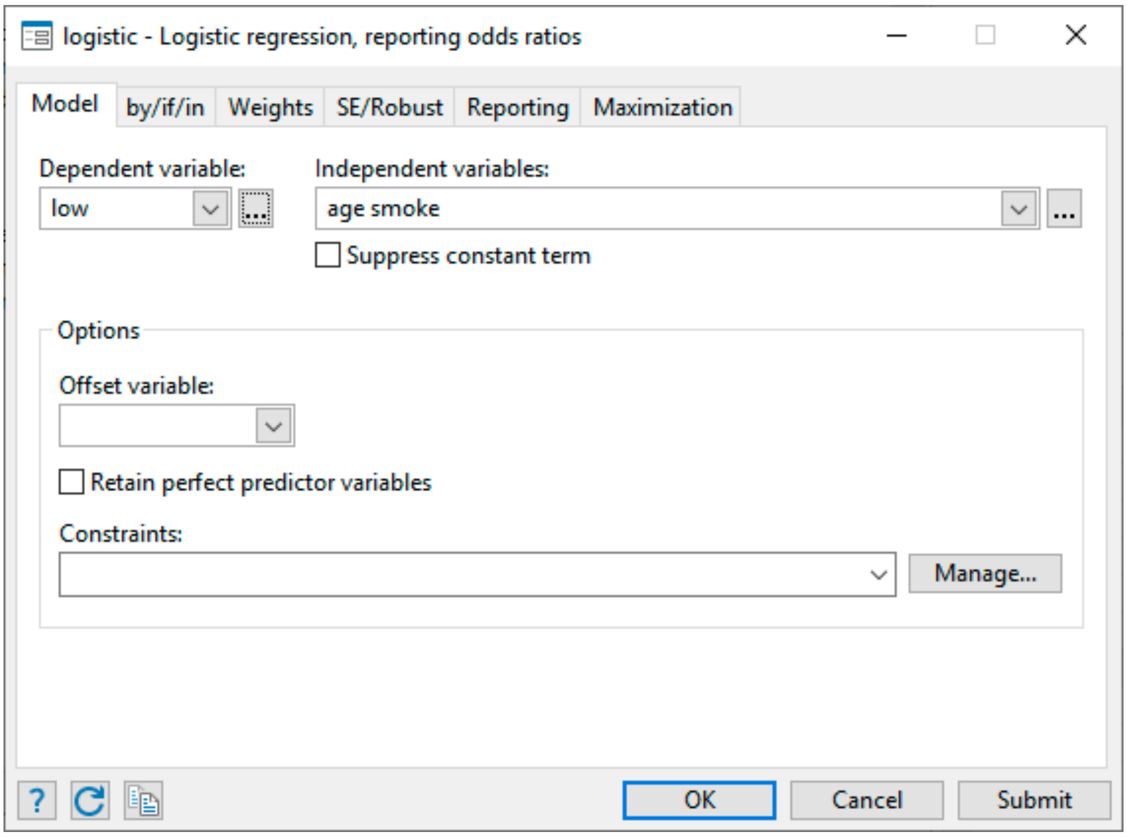
Options

(●) Standard display
( ) Display additional statistics
( ) No display; just calculate mean
[ ] Use variable's display format
5 Separator line every N variables (set 0 for none)

```
. summarize low age smoke

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
         low |        189    .3121693    .4646093          0          1
         age |        189     23.2381    5.298678         14         45
       smoke |        189    .3915344    .4893898          0          1
```

Stata shows us the command that we could have typed in command mode—`summarize low age smoke`—before displaying the results of our request.

Next we fit a logistic regression model of `low` on `age` and `smoke`. We select **Statistics** > **Binary outcomes** > **Logistic regression**, fill in the fields, and click on **OK**.



Dependent variable: low

Independent variables: age smoke

Suppress constant term

Offset variable:

Retain perfect predictor variables

Constraints:

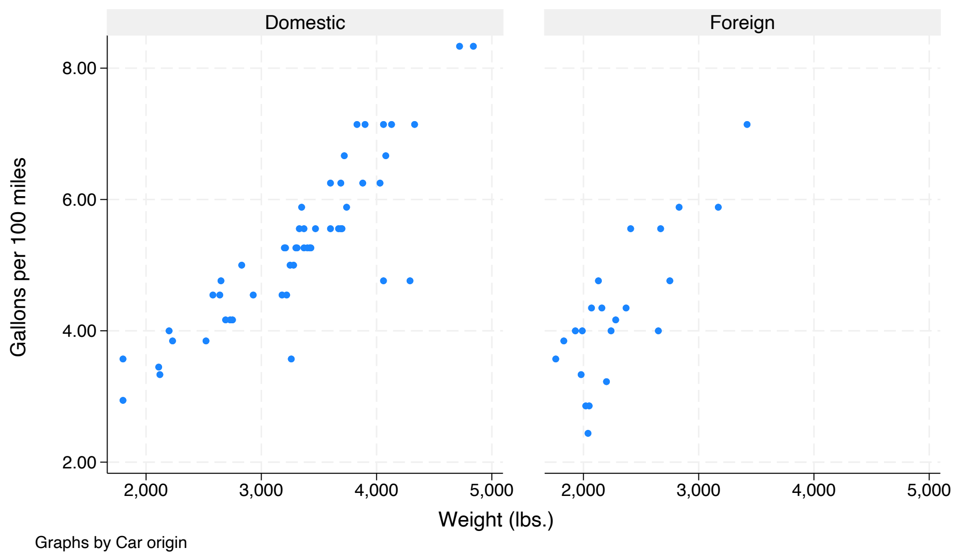
```
. logistic low age smoke

Logistic regression                                     Number of obs =    189
                                                        LR chi2(2)    =   7.40
                                                        Prob > chi2   = 0.0248
Log likelihood = -113.63815                             Pseudo R2     = 0.0315

------------------------------------------------------------------------------
         low | Odds ratio   Std. err.      z    P>|z|     [95% conf. interval]
-------------+----------------------------------------------------------------
         age |   .9514394   .0304194   -1.56   0.119     .8936482    1.012968
       smoke |   1.997405    .642777    2.15   0.032     1.063027    3.753081
       _cons |   1.062798   .8048781    0.08   0.936     .2408901    4.689025
------------------------------------------------------------------------------
Note: _cons estimates baseline odds.
```

Here is an extract of a Stata session using the command language:

```
. use https://www.stata-press.com/data/r19/auto
(1978 automobile data)

. summarize mpg weight

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
         mpg |         74     21.2973    5.785503         12         41
      weight |         74    3019.459    777.1936       1760       4840
```

The user typed `summarize mpg weight` and Stata responded with a table of summary statistics. Other commands would produce different results:

```
. generate gp100m = 100/mpg

. label var gp100m "Gallons per 100 miles"

. format gp100m %5.2f

. correlate gp100m weight
(obs=74)

             |   gp100m   weight
-------------+------------------
      gp100m |   1.0000
      weight |   0.8544   1.0000

. regress gp100m weight gear_ratio

      Source |       SS           df       MS      Number of obs   =        74
-------------+----------------------------------   F(2, 71)        =     96.65
       Model |  87.4543721         2  43.7271861   Prob > F        =    0.0000
    Residual |  32.1218886        71  .452420967   R-squared       =    0.7314
-------------+----------------------------------   Adj R-squared   =    0.7238
       Total |  119.576261        73  1.63803097   Root MSE        =    .67262

------------------------------------------------------------------------------
      gp100m | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
      weight |   .0014769   .0001556     9.49   0.000     .0011665    .0017872
  gear_ratio |   .1566091   .2651131     0.59   0.557    -.3720115    .6852297
       _cons |   .0878243   1.198434     0.07   0.942    -2.301786    2.477435
------------------------------------------------------------------------------

. scatter gp100m weight, by(foreign)
```



The user-interface model is type a little, get a little, etc., so that the user is always in control.

Stata's model for a dataset is that of a table—the rows are the observations and the columns are the variables:

```
. list mpg weight gp100m in 1/10

     ┌─────────────────────────┐
     │ mpg   weight   gp100m   │
     ├─────────────────────────┤
  1. │  22    2,930     4.55   │
  2. │  17    3,350     5.88   │
  3. │  22    2,640     4.55   │
  4. │  20    3,250     5.00   │
  5. │  15    4,080     6.67   │
     ├─────────────────────────┤
  6. │  18    3,670     5.56   │
  7. │  26    2,230     3.85   │
  8. │  20    3,280     5.00   │
  9. │  16    3,880     6.25   │
 10. │  19    3,400     5.26   │
     └─────────────────────────┘
```

Observations are numbered; variables are named.

Stata is fast. That speed is due partly to careful programming, and partly because Stata keeps the data in memory. Stata's file model is that of a word processor: a dataset may exist on disk, but the dataset in memory is a copy. Datasets are loaded into memory, where they are worked on, analyzed, changed, and then perhaps stored back on disk.

Working on a copy of the data in memory makes Stata safe for interactive use. The only way to harm the permanent copy of your data on disk is if you explicitly save over it.

Having the data in memory means that the dataset size is limited by the amount of computer memory. Stata stores the data in memory in an efficient format—you will be surprised how much data can fit. Nevertheless, if you work with extremely large datasets, you may run into memory constraints. You will want to learn how to store your data as efficiently as possible; see [D] **compress**.

3 Resources for learning and using Stata

Contents

3.1 Overview

The *Getting Started* manual, *User's Guide*, and *Reference* manuals are the primary tools for learning about Stata; however, there are many other sources of information. A few are listed below.

- Stata itself. Stata has a `search` command that makes it easy search a topic to find and to execute a Stata command. See **[U] 4 Stata's help and search facilities**.
- The Stata website. Visit https://www.stata.com. Much of the site is dedicated to user support; see **[U] 3.2.1 The Stata website (www.stata.com)**.
- The Stata YouTube Channel. Visit https://www.youtube.com/user/statacorp/. The site is regularly updated with video demonstrations of Stata.

- *The Stata Blog*, 𝕏, and Facebook. Visit https://blog.stata.com, https://x.com/stata/, and https://www.facebook.com/statacorp/. See **[U] 3.2.3 The Stata Blog: Not Elsewhere Classified** and **[U] 3.2.5 Stata on social media**.
- The Stata Press website. Visit https://www.stata-press.com. This site contains the datasets used throughout the Stata manuals; see **[U] 3.3 Stata Press**.
- The Stata Forum. An active group of Stata users communicate over an internet forum; see **[U] 3.2.4 The Stata Forum**.
- The *Stata Journal*. The *Stata Journal* contains reviewed papers, regular columns, book reviews, and other material of interest to researchers applying statistics in a variety of disciplines. See **[U] 3.4 The Stata Journal**.
- The Stata software distribution site and other user-provided software distribution sites. Stata itself can download and install updates and additions. We provide official updates to Stata—type `update query` or select **Help > Check for updates**. We also provide community-contributed additions to Stata and links to other user-provided sites—type `net` or select **Help > SJ and community-contributed features**; see **[U] 3.5 Updating and adding features from the web**.
- NetCourses. We offer training via the internet. Details are in **[U] 3.6.2 NetCourses**.
- Classroom training courses. We offer in-depth training courses at third-party sites around the United States. Details are in **[U] 3.6.3 Classroom training courses**.
- Web-based training courses. We offer the same content from our classroom training over the web. Details are in **[U] 3.6.4 Web-based training courses**.
- Organizational training courses. We offer both in-person and virtual customized training for your institution. Details are in **[U] 3.6.5 Organizational training courses**.
- Webinars. We offer free, short online webinars to learn about Stata from our experts. Details are in **[U] 3.6.6 Webinars**.
- Books and support materials. Supplementary Stata materials are available; see **[U] 3.7 Books and other support materials**.
- Technical support. We provide technical support by email and telephone; see **[U] 3.8 Technical support**.

3.2 Stata on the internet (www.stata.com and other resources)

3.2.1 The Stata website (www.stata.com)

Point your browser to https://www.stata.com and click on **Support**. More than half our website is dedicated to providing support to users.

- The website provides answers to FAQs (frequently asked questions) on Windows, Mac, Unix, statistics, programming, Mata, internet capabilities, graphics, and data management. These FAQs run the gamut from “I cannot save/open files” to “What does ‘completely determined’ mean in my logistic regression output?” Most users will find something of interest.
- The website provides detailed information about NetCourses, along with the current schedule; see **[U] 3.6.2 NetCourses**.

- The website provides information about Stata courses and meetings, both in the United States and elsewhere. See **[U] 3.6.1 Conferences and users group meetings**, **[U] 3.6.3 Classroom training courses**, **[U] 3.6.4 Web-based training courses**, and **[U] 3.6.5 Organizational training courses**.
- The website provides an online bookstore for Stata-related books and other supplementary materials; see **[U] 3.7 Books and other support materials**.
- The website provides links to information about statistics: other statistical software providers, book publishers, statistical journals, statistical organizations, and statistical listservers.
- The website provides links to resources for learning Stata at https://www.stata.com/links/resources-for-learning-stata. Be sure to look at these materials, as many outstanding resources about Stata are listed here.

In short, the website provides up-to-date information on all support materials and, where possible, provides the materials themselves. Visit https://www.stata.com if you can.

3.2.2 The Stata YouTube Channel

Visit Stata's YouTube Channel at https://www.youtube.com/user/statacorp/ to view video demonstrations on a wide variety of topics ranging from basic data management and graphics to more advanced statistical analyses, such as ANOVA, regression, and SEM. New demonstrations are regularly added.

3.2.3 The Stata Blog: Not Elsewhere Classified

Stata's official blog can be found at https://blog.stata.com and contains news and advice related to the use of Stata. The articles appearing in the blog are individually signed and are written by the same people who develop, support, and sell Stata.

3.2.4 The Stata Forum

Statalist is a forum dedicated to Stata, where thousands of Stata users discuss Stata and statistics. It is run and moderated by Stata users and maintained by StataCorp. Statalist has a long history of high-quality discussion dating back to 1994.

Many knowledgeable users are active on the forum, as are the StataCorp technical staff. Anyone may join, and new-to-Stata members are welcome. Instructions for joining can be found at https://www.statalist.org. Register and participate, or simply lurk and read the discussions.

Before posting a question to Statalist, you will want to read the Statalist FAQ, which can be found at https://www.statalist.org/forums/help/.

3.2.5 Stata on social media

StataCorp has an official presence on 𝕏, Facebook, Instagram, and LinkedIn. You can follow us on 𝕏 at https://x.com/stata/. You find us on Facebook and Instagram at https://www.facebook.com/statacorp/ and https://www.instagram.com/statacorp/. Connect with us on LinkedIn at https://www.linkedin.com/company/statacorp/. These are good ways to stay up-to-the-minute with the latest Stata information.

3.2.6 Other internet resources on Stata

Many other people have published information on the internet about Stata such as tutorials, examples, and datasets. Visit https://www.stata.com/links/ to explore other Stata and statistics resources on the internet.

3.3 Stata Press

Stata Press is the publishing arm of StataCorp LLC and publishes books, manuals, and journals about Stata statistical software and about general statistics topics for professional researchers of all disciplines.

Point your browser to https://www.stata-press.com. This site is devoted to the publications and activities of Stata Press.

- Datasets that are used in the Stata *Reference* manuals and other books published by Stata Press may be downloaded. Visit https://www.stata-press.com/data/. These datasets can be used in Stata by simply typing `use https://www.stata-press.com/data/r19/`*dataset_name*; for example, type `use https://www.stata-press.com/data/r19/auto`. You could also type `webuse auto`; see [D] **webuse**.
- An online catalog of all our books and multimedia products is at https://www.stata-press.com/books/. We have tried to include enough information, such as table of contents and preface material, so that you may tell whether the book is appropriate for you.
- Information about forthcoming publications is posted at https://www.stata-press.com/forthcoming/.

3.4 The Stata Journal

The *Stata Journal* (SJ) is a printed and electronic journal, published quarterly, containing articles about statistics, data analysis, teaching methods, and effective use of Stata's language. The SJ publishes reviewed papers together with shorter notes and comments, regular columns, tips, book reviews, and other material of interest to researchers applying statistics in a variety of disciplines. The SJ is a publication for all Stata users, both novice and experienced, with different levels of expertise in statistics, research design, data management, graphics, reporting of results, and in Stata, in particular.

The SJ is published by and available from SAGE Publishing. Tables of contents for past issues and abstracts of the articles are available at https://www.stata-journal.com/archives/. PDF copies of articles published at least three years ago are available for free from SAGE Publishing's SJ webpage.

We recommend that all users subscribe to the SJ. Visit https://www.stata-journal.com to learn more about the SJ. Subscription information is available at https://www.stata-journal.com/subscription.

To obtain any programs associated with articles in the SJ, type

```
. net from https://www.stata-journal.com/software
```

or

- Select **Help > SJ and community-contributed features**
- Click on **Stata Journal**

3.5 Updating and adding features from the web

Stata itself can open files on the internet. Stata understands http, https, and ftp protocols.

First, try this:

```
. use https://www.stata.com/manual/oddeven, clear
```

That will load an uninteresting dataset into your computer from our website. If you have a home page, you can use this feature to share datasets with coworkers. Save a dataset on your home page, and researchers worldwide can use it. See [R] **net**.

3.5.1 Official updates

Although we follow no formal schedule for the release of updates, we typically provide updates to Stata approximately once a month. Installing the updates is easy. Type

```
. update query
```

or select **Help > Check for updates**. Do not be concerned; nothing will be installed unless and until you say so. Once you have installed the update, you can type

```
. help whatsnew
```

or select **Help > What's new?** to find out what has changed. We distribute official updates to fix bugs and to add new features.

3.5.2 Unofficial updates

There are also "unofficial" updates—additions to Stata written by Stata users, which includes members of the StataCorp technical staff. Stata is programmable, and even if you never write a Stata program, you may find these additions useful, some of them spectacularly so. Start by typing

```
. net from https://www.stata.com
```

or select **Help > SJ and community-contributed features**.

Be sure to visit the Statistical Software Components (SSC) Archive, which hosts a large collection of free additions to Stata. The `ssc` command makes it easy for you to find, install, and uninstall packages from the SSC Archive. Type

```
. ssc whatsnew
```

to find out what's new at the site. If you find something that interests you, type

```
. ssc describe pkgname
```

for more information. If you have already installed a package, you can check for and optionally install updates by typing

```
. ado update pkgname
```

To check for and optionally install updates to all the packages you have previously installed, type

```
. ado update all
```

See [U] **29 Using the internet to keep up to date**.

3.6 Conferences and training

3.6.1 Conferences and users group meetings

Every year, users around the world meet in a variety of locations to discuss Stata software and to network with other users. At these meetings, users and StataCorp developers alike share new features, novel Stata uses, and best practices. StataCorp organizes and hosts the Stata Conference in both the United States and Canada, as well as supports meetings worldwide.

Visit https://www.stata.com/meeting/ for a list of upcoming conferences and meetings.

3.6.2 NetCourses

We offer courses on Stata at both introductory and advanced levels. Courses on software are typically expensive and time consuming. They are expensive because, in addition to the direct costs of the course, participants must travel to the course site. Courses over the internet save everyone time and money.

We offer courses over the internet and call them Stata NetCourses.

- **What is a NetCourse?**
  A NetCourse is a course offered through the Stata website that varies in length from 7 to 8 weeks. Everyone with an email address and a web browser can participate.
- **How does it work?**
  Every Friday a lesson is posted on a password-protected website. After reading the lesson over the weekend or perhaps on Monday, participants then post questions and comments on a message board. Course leaders typically respond to the questions and comments on the same day they are posted. Other participants are encouraged to amplify or otherwise respond to the questions or comments as well. The next lesson is then posted on Friday, and the process repeats.
- **How much of my time does it take?**
  It depends on the course, but the introductory courses are designed to take roughly 3 hours per week.
- **There are three of us here—can just one of us enroll and then redistribute the NetCourse materials ourselves?**
  We ask that you not. NetCourses are priced to cover the substantial time input of the course leaders. Moreover, enrollment is typically limited to prevent the discussion from becoming unmanageable. The value of a NetCourse, just like a real course, is the interaction of the participants, both with each other and with the course leaders.
- **I've never taken a course by internet before. I can see that it might work, but then again, it might not. How do I know I will benefit?**
  All Stata NetCourses come with a 30-day satisfaction guarantee. The 30 days begins after the conclusion of the final lesson.

You can learn more about the current NetCourse offerings by visiting https://www.stata.com/netcourse/.

NetCourseNow

A NetCourseNow offers the same material as NetCourses but it allows you to choose the time and pace of the course, and you have a personal NetCourse instructor.

- **What is a NetCourseNow?**
  A NetCourseNow offers the same material as a NetCourse, but allows you to move at your own pace and to specify a starting date. With a NetCourseNow, you also have the added benefit of a personal NetCourse instructor whom you can email directly with questions about lessons and exercises. You must have an email address and a web browser to participate.
- **How does it work?**
  All course lessons and exercises are posted at once, and you are free to study at your own pace. You will be provided with the email address of your personal NetCourse instructor to contact when you have questions.
- **How much of my time does it take?**
  A NetCourseNow allows you to set your own pace. How long the course takes and how much time you spend per week is up to you.

3.6.3 Classroom training courses

Classroom training courses are intensive, in-depth courses that will teach you to use Stata or, more specifically, to use one of Stata's advanced statistical procedures. Courses are taught by StataCorp at third-party sites around the United States.

- **How is a classroom training course taught?**
  These are interactive, hands-on sessions. Participants work along with the instructor so that they can see firsthand how to use Stata. Questions are encouraged.
- **Do I need my own computer?**
  Because the sessions are in computer labs running the latest version of Stata, there is no need to bring your own computer. Of course, you may bring your own computer if you have a registered copy of Stata you can use.
- **Do I get any notes?**
  You get a complete set of printed notes for each class, which includes not only the materials from the lessons but also all the output from the example commands.

See https://www.stata.com/training/classroom-and-web/ for all course offerings.

3.6.4 Web-based training courses

Web-based training courses, like classroom training courses, are intensive, in-depth courses that will teach you to use Stata or, more specifically, to use one of Stata's advanced statistical procedures. Courses are taught by StataCorp, and you join the course online from your home or office.

- **How is a web-based training course taught?**
  These are interactive, hands-on sessions. Participants work along with the instructor so that they can see firsthand how to use Stata. Questions are encouraged.
- **Do I need my own computer and Stata license?**
  You will need a computer with a high-speed internet connection to join the course and to run Stata. If you do not have a license for the current version of Stata, you will be provided with a temporary license.
- **Do I get any notes?**
  You get a complete set of notes for each class, which includes not only the materials from the lessons but also all the output from the example commands.

See https://www.stata.com/training/classroom-and-web/ for all course offerings.

3.6.5 Organizational training courses

Organizational training courses are courses that are tailored to the needs of each institution. StataCorp personnel can come to your site to teach what you need, whether it be to teach new users or to show how to use a specialized tool in Stata. This training is also available virtually.

- **How is an organizational training course taught?**
  These are interactive, hands-on sessions, just like our classroom training courses. You will need a computer for each participant.
- **What topics are available?**
  We offer training in anything and everything related to Stata. You work with us to put together a curriculum that matches your needs.
- **How does licensing work?**
  We will supply you with the licenses you need for the training session, whether the training is in a lab or for individuals working on laptops. We will send the licensing and installation instructions so that you can have everything up and running before the session starts.

See https://www.stata.com/training/organizational/ for all the details.

3.6.6 Webinars

Webinars are free, live demonstrations of Stata features for both new and experienced Stata users. The *Ready. Set. Go Stata.* webinar shows new users how to quickly get started manipulating, graphing, and analyzing data. Already familiar with Stata? Discover a few of our developers' favorite features of Stata in our *Tips and tricks* webinar. The one-hour specialized feature webinars provide both new and experienced users with an in-depth look at one of Stata's statistical, graphical, data management, or reporting features.

- **How do I access the webinar?**
  Webinars are given live using either Adobe Connect software or Zoom.
- **Do I need my own computer and Stata license?**
  You will need a computer with high-speed internet connection to join the webinar and to run Adobe Connect or Zoom. You do not need access to Stata to attend.
- **What is the cost to attend?**
  Webinars are free, but you must register to attend. Registrations are limited so we recommend registering early.

See https://www.stata.com/training/webinar/ for all the details.

3.7 Books and other support materials

3.7.1 For readers

There are books published about Stata, both by us and by others. Visit the Stata Bookstore at https://www.stata.com/bookstore/. We include the table of contents and comments written by a member of our technical staff, explaining why we think this book might interest you.

3.7.2 For authors

If you have written a book related to Stata and would like us to consider adding it to our bookstore, email bookstore@stata.com.

If you are writing a book, join our free Author Support Program. Stata professionals are available to review your Stata code to ensure that it is efficient and reflects modern usage, production specialists are available to help format Stata output, and editors and statisticians are available to ensure the accuracy of Stata-related content. Visit https://www.stata.com/authorsupport/.

If you are thinking about writing a Stata-related book, consider publishing it with Stata Press. Email editor@stata-press.com.

3.7.3 For editors

If you are editing a book that demonstrates Stata usage and output, join our free Editor Support program. Stata professionals are available to review the Stata content of book proposals, review Stata code and ensure output is efficient and reflects modern usage, provide advice about formatting of Stata output (including graphs), and review the accuracy of Stata-related content. Visit https://www.stata.com/publications/editor-support-program/.

3.7.4 For instructors

Teaching your course with Stata provides your students with tools and skills that translate to their professional life. Our teaching resources page provides access to resources for instructors, including links to our video tutorials, *Ready. Set. Go Stata.* webinar, Stata cheat sheets, and more. Visit https://www.stata.com/teaching-with-stata/.

3.8 Technical support

We are committed to providing superior technical support for Stata software. To assist you as efficiently as possible, please follow the procedures listed below.

3.8.1 Register your software

You must register your software to be eligible for technical support, updates, special offers, and other benefits. By registering, you will receive the *Stata News*, and you may access our support staff for free with any question that you encounter. You may register your software electronically.

After installing Stata and successfully entering your License and Activation Key, your default web browser will open to the online registration form at the Stata website. You may also manually point your web browser to https://www.stata.com/register/ if you wish to register your copy of Stata at a later time.

3.8.2 Before contacting technical support

Before you spend the time gathering the information our technical support department needs, make sure that the answer does not already exist in the help files. You can use the `help` and `search` commands to find all the entries in Stata that address a given subject. Be sure to try selecting
Help > Contents. Check the manual for a particular command. There are often examples that address questions and concerns. Another good source of information is our website. You should keep a bookmark to our frequently asked questions page (https://www.stata.com/support/faqs/).

If you do need to contact technical support, visit https://www.stata.com/support/tech-support/ for more information.

3.8.3 Technical support by email

This is the preferred method of asking a technical support question. It has the following advantages:

- You will receive a prompt response from us saying that we have received your question and that it has been forwarded to *Technical Services* to answer.
- We can route your question to a specialist for your particular question.
- Questions submitted via email may be answered after normal business hours, or even on weekends or holidays. Although we cannot promise that this will happen, it may, and your email inquiry is bound to receive a faster response than leaving a message on Stata's voicemail.
- If you are receiving an error message or an unexpected result, it is easy to include a log file that demonstrates the problem.

Please visit https://www.stata.com/support/tech-support/ for information about contacting technical support.

3.8.4 Technical support by phone

Our installation support telephone number is 979-696-4600. Please have your serial number handy. It is also best if you are at your computer when you call. Telephone support is reserved for installation questions. If your question does not involve installation, the question should be submitted via email.

Visit https://www.stata.com/support/tech-support/ for information about contacting technical support.

3.8.5 Comments and suggestions for our technical staff

By all means, send in your comments and suggestions. Your input is what determines the changes that occur in Stata between releases, so if we do not hear from you, we may not include your most desired new feature! Email is preferred, as this provides us with a permanent copy of your request. When requesting new features, please include any references that you would like us to review should we develop those new features. Email your suggestions to service@stata.com.

4 Stata's help and search facilities

Contents

4.1 Introduction

To access Stata's help, you will either

1. select **Help** from the menus, or
2. use the `help` and `search` commands.

Regardless of the method you use, results will be shown in the Viewer or Results windows. Blue text indicates a hypertext link, so you can click to go to related entries.

4.2 Getting started

The first time you use help, try one of the following:

1. select **Help** > **Advice** from the menu bar, or
2. type `help advice`.

Either step will open the `help_advice` help file within a Viewer window. The advice file provides you with steps to search Stata to find information on topics and commands that interest you.

4.3 help: Stata's help system

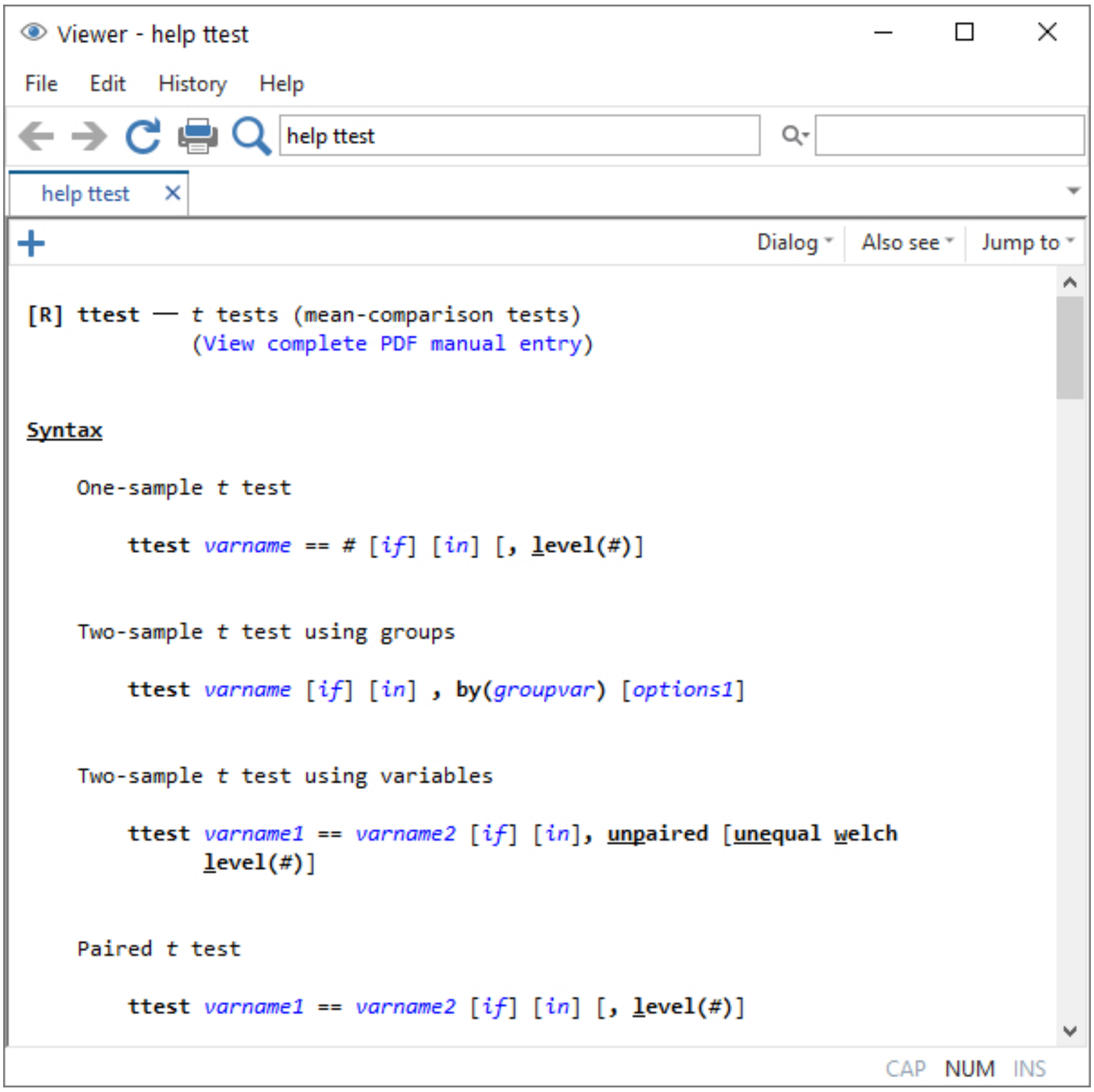
When you

1. Select **Help** > **Stata command...**
   Type a command name in the Command edit field
   Click on OK, or
2. Type `help` followed by a command name

you access Stata's help files. These files provide shortened versions of what is in the printed manuals. Let's access the help file for Stata's `ttest` command. Do one of the following:

1. Select **Help > Stata command...**
   Type `ttest` in the Command edit field
   Click on OK, or
2. Type `help ttest`

Regardless of which you do, the result will be



```
[R] ttest — t tests (mean-comparison tests)
            (View complete PDF manual entry)


Syntax

    One-sample t test

        ttest varname == # [if] [in] [, level(#)]


    Two-sample t test using groups

        ttest varname [if] [in] , by(groupvar) [options1]


    Two-sample t test using variables

        ttest varname1 == varname2 [if] [in], unpaired [unequal welch
              level(#)]


    Paired t test

        ttest varname1 == varname2 [if] [in] [, level(#)]
```

The trick is in already knowing that Stata's command for testing equality of means is `ttest` and not, say, `meanstest`. The solution to that problem is searching.

4.4 Accessing PDF manuals from help entries

Every help file in Stata links to the equivalent manual entry. If you are reading `help ttest`, simply click on `(View complete PDF manual entry)` below the title to go directly to the [R] **ttest** manual entry.

We provide some tips for viewing Stata's PDF documentation at https://www.stata.com/support/faqs/resources/pdf-documentation-tips/.

4.5 Searching

If you do not know the name of the Stata command you are looking for, you can search for it by keyword,

1. Select **Help > Search...**
   Type keywords in the edit field
   Click on OK
2. Type `search` followed by the keywords

`search` matches the keywords you specify to a database and returns matches found in Stata commands, FAQs at www.stata.com, official blogs, and articles that have appeared in the *Stata Journal*. It can also find community-contributed additions to Stata available over the web.

`search` does a better job when what you want is based on terms commonly used or when what you are looking for might not already be installed on your computer.

4.6 More on search

However you access `search`—command or menu—it does the same thing. You tell `search` what you want information about, and it searches for relevant entries. By default, `search` looks for the topic across all sources, including the system help, the FAQs at the Stata website, the *Stata Journal*, and all Stata-related internet sources including community-contributed additions.

`search` can be used broadly or narrowly. For instance, if you want to perform the Kolmogorov–Smirnov test for equality of distributions, you could type

```
. search Kolmogorov-Smirnov test of equality of distributions
[R]     ksmirnov . . . . . . Kolmogorov-Smirnov equality of distributions test
        (help ksmirnov)
```

In fact, we did not have to be nearly so complete—typing `search Kolmogorov-Smirnov` would have been adequate. Had we specified our request more broadly—looking up `equality of distributions`—we would have obtained a longer list that included `ksmirnov`.

Here are guidelines for using `search`.

- Capitalization does not matter. Look up `Kolmogorov-Smirnov` or `kolmogorov-smirnov`.
- Punctuation does not matter. Look up `kolmogorov smirnov`.
- Order of words does not matter. Look up `smirnov kolmogorov`.
- You may abbreviate, but how much depends. Break at syllables. Look up `kol smir`. `search` tends to tolerate a lot of abbreviation; it is better to abbreviate than to misspell.
- The words a, an, and, are, for, into, of, on, to, the, and with are ignored. Use them—look up `equality of distributions`—or omit them—look up `equality distributions`—it makes no difference.
- `search` tolerates plurals, especially when they can be formed by adding an *s*. Even so, it is better to look up the singular. Look up `normal distribution`, not `normal distributions`.
- Specify the search criterion in English, not in computer jargon.
- Use American spellings. Look up `color`, not `colour`.
- Use nouns. Do not use -ing words or other verbs. Look up `median tests`, not `testing medians`.

- Use few words. Every word specified further restricts the search. Look up `distribution`, and you get one list; look up `normal distribution`, and the list is a sublist of that.
- Sometimes words have more than one context. The following words can be used to restrict the context:

  a. `data`, meaning in the context of data management. Order could refer to the order of data or to order statistics. Look up `order data` to restrict order to its data management sense.

  b. `statistics` (abbreviation `stat`), meaning in the context of statistics.
  Look up `order statistics` to restrict order to the statistical sense.

  c. `graph` or `graphs`, meaning in the context of statistical graphics. Look up `median graphs` to restrict the list to commands for graphing medians.

  d. `utility` (abbreviation `util`), meaning in the context of utility commands. The `search` command itself is not data management, not statistics, and not graphics; it is a utility.

  e. `programs` or `programming` (abbreviation `prog`), to mean in the context of programming. Look up `programming scalar` to obtain a sublist of scalars in programming.

`search` has other features, as well; see **[U] 4.8 search: All the details**.

4.7 More on help

Both `help` and `search` are understanding of some mistakes. For instance, you may abbreviate some command names. If you type either `help regres` or `help regress`, you will bring up the help file for `regress`.

When `help` cannot find the command you are looking for among Stata's official help files or any community-contributed additions you have installed, Stata automatically performs a search. For instance, typing `help ranktest` causes Stata to reply with "help for ranktest not found", and then Stata performs `search ranktest`. The search tells you that `ranktest` is available in the *Enhanced routines for IV/GMM estimation and testing* article in Stata Journal, Volume 7, Number 4.

Stata can run into some problems with abbreviations. For instance, Stata has a command with the inelegant name `ksmirnov`. You forget and think the command is called `ksmir`:

```
. help ksmir
No entries found for search on "ksmir"
```

A help file for `ksmir` was not found, so Stata automatically performed a search on the word. The message indicates that a search of `ksmir` also produced no results. You should type `search` followed by what you are really looking for: `search kolmogorov smirnov`.

4.8 search: All the details

The `search` command actually provides a few features that are not available from the **Help** menu. The full syntax of the `search` command is

`search` *word* [*word* ...] [, [`all` | `local` | `net`] `author` `entry` `exact` `faq`

  `historical` or `manual` `sj`]

where underlining indicates the minimum allowable abbreviation and [`brackets`] indicate optional.

`all`, the default, specifies that the search be performed across both the local keyword database and the net materials.

`local` specifies that the search be performed using only Stata's keyword database.

`net` specifies that the search be performed across the materials available via Stata's `net` command. Using `search` `word` [*word* ...], `net` is equivalent to typing `net search word` [*word* ...] (without options); see [R] **net**.

`author` specifies that the search be performed on the basis of author's name rather than keywords.

`entry` specifies that the search be performed on the basis of entry IDs rather than keywords.

`exact` prevents matching on abbreviations.

`faq` limits the search to the FAQs posted on the Stata and other select websites.

`historical` adds to the search entries that are of historical interest only. By default, such entries are not listed.

`or` specifies that an entry be listed if any of the words typed after `search` are associated with the entry. The default is to list the entry only if all the words specified are associated with the entry.

`manual` limits the search to entries in the *User's Guide* and all the *Reference* manuals.

`sj` limits the search to entries in the *Stata Journal*.

4.8.1 How search works

`search` has a database—files—containing the titles, etc., of every entry in the *User's Guide*, *Reference* manuals, undocumented help files, NetCourses, Stata Press books, FAQs posted on the Stata website, videos on StataCorp's YouTube channel, selected articles on StataCorp's official blog, selected community-contributed FAQs and examples, and the articles in the *Stata Journal*. In this file is a list of words associated with each entry, called keywords.

When you type `search` *xyz*, `search` reads this file and compares the list of keywords with *xyz*. If it finds *xyz* in the list or a keyword that allows an abbreviation of *xyz*, it displays the entry.

When you type `search` *xyz abc*, `search` does the same thing but displays an entry only if it contains both keywords. The order does not matter, so you can `search linear regression` or `search regression linear`.

How many entries `search` finds depends on how the search database was constructed. We have included a plethora of keywords under the theory that, for a given request, it is better to list too much rather than risk listing nothing at all. Still, you are in the position of guessing the keywords. Do you look up normality test, normality tests, or tests of normality? Normality test would be best, but all would work. In general, use the singular and strike the unnecessary words. We provide guidelines for specifying keywords in [U] **4.6 More on search** above.

4.8.2 Author searches

`search` ordinarily compares the words following `search` with the keywords for the entry. If you specify the `author` option, however, it compares the words with the author's name. In the search database, we have filled in author names for *Stata Journal* articles, Stata Press books, StataCorp's official blog, and FAQs.

For instance, in the *Acknowledgments* of [R] **kdensity**, you will discover the name Isaías H. Salgado-Ugarte. You want to know if he has written any articles in the *Stata Journal*. To find out, type

```
. search Salgado-Ugarte, author
  (output omitted)
```

Names like Salgado-Ugarte are confusing to some people. `search` does not require you specify the entire name; what you type is compared with each "word" of the name, and if any part matches, the entry is listed. The hyphen is a special character, and you can omit it. Thus you can obtain the same list by looking up Salgado, Ugarte, or Salgado Ugarte without the hyphen.

4.8.3 Entry ID searches

If you specify the `entry` option, `search` compares what you have typed with the entry ID. The entry ID is not the title—it is the reference listed to the left of the title that tells you where to look. For instance, in

```
         regress  . . . . . . . . . . . . . . . . . . . . . . Linear regression
            (help regress)
```

"[R] regress" is the entry ID. In

```
GS       . . . . . . . . . . . . . . . . . . . . . . . . Getting Started manual
```

"GS" is the entry ID. In

```
SJ-14-4  gr0059  . . . . . Plotting regression coefficients and other estimates
         (help coefplot if installed)  . . . . . . . . . . . . . . .   B. Jann
         Q4/14   SJ 14(4):708--737
         alternative to marginsplot that plots results from any
         estimation command and combines results from several models
         into one graph
```

"SJ-14-4 gr0059" is the entry ID.

`search` with the `entry` option searches these entry IDs.

Thus you could generate a table of contents for the *Reference* manuals by typing

```
. search [R], entry
  (output omitted)
```

4.8.4 FAQ searches

To search across the FAQs, specify the `faq` option:

```
. search logistic regression, faq
  (output omitted)
```

4.8.5 Return codes

In addition to indexing the entries in the *User's Guide* and all the *Stata Reference* manuals, `search` also can be used to look up return codes.

To see information about return code 131, type

```
. search rc 131
[R]     error messages  . . . . . . . . . . . . . . . . . . . .  Return code 131
        not possible with test;
        You requested a test of a hypothesis that is nonlinear in the
        variables.  test tests only linear hypotheses.  Use testnl.
```

To get a list of all Stata return codes, type

```
. search rc
  (output omitted)
```

4.9 net search: Searching net resources

When you select **Help > Search...**, there are two types of searches to choose. The first, which has been discussed in the previous sections, is to **Search documentation and FAQs**. The second is to **Search net resources**. This feature of Stata searches resources over the internet.

When you choose **Search net resources** in the search dialog box and enter *keywords* in the field, Stata searches all community-contributed programs on the internet, including community-contributed additions published in the *Stata Journal*. The results are displayed in the Viewer, and you can click to go to any of the matches found.

Equivalently, you can type `net search` *keywords* on the Stata command line to display the results in the Results window. For the full syntax for using the `net search` command, see [R] **net search**.

5 Editions of Stata

Contents

5.1 StataNow

First and foremost, StataNow is Stata. It is a continuous-release version of Stata that offers new features as soon as they are ready. StataNow is the result of our ongoing effort to deliver the best Stata—the most current Stata—to our users.

StataNow provides access to new features as soon as they are certified ready by the development team. Those features are added to StataNow via free updates as soon as they are available. Thus, StataNow provides access to new features sooner. For instance, StataNow 19 contains features that will also be part of a future major release, Stata 20. View the list of the latest StataNow features at https://www.stata.com/new-in-stata/features/#statanow.

The features in StataNow are fully tested, fully certified, well documented, and version controlled (if needed), as well as polished to our customary high quality. These features are prioritized in the development cycle to be released as soon as they are ready so that users can take advantage of them right away. As always, all versions of Stata are updated regularly with any corrections and necessary improvements.

The new features in StataNow are released continuously throughout the current release until the next major release. They are not released according to any preset schedule. All StataNow features are marked as such throughout the Stata documentation and the Stata website.

Because StataNow is Stata, when we mention "Stata" throughout our documentation and website, we also mean "StataNow". We will be specific about StataNow for features available only in StataNow. And because StataNow is Stata, it is available in all editions (StataNow/MP, StataNow/SE, and StataNow/BE) and on all supported platforms (Windows, Mac, and Linux). Throughout the documentation and website, we will usually refer to just Stata/MP, Stata/SE, and Stata/BE for simplicity. If you have a StataNow license, you can take this to mean StataNow/MP, StataNow/SE, and StataNow/BE.

For more information, see https://www.stata.com/statanow/.

5.2 Platforms

Stata is available for a variety of systems, including

Stata for Windows, 64-bit x86-64

Stata for Mac, 64-bit x86-64

Stata for Linux, 64-bit x86-64

Which version of Stata you run does not matter—Stata is Stata. You instruct Stata in the same way and Stata produces the same results, right down to the random-number generator. Even files can be shared. A dataset created on one computer can be used on any other computer, and the same goes for graphs, programs, or any file Stata uses or produces. Moving files across platforms is simply a matter of copying them; no translation is required.

Some computers, however, are faster than others. Some computers have more memory than others. Computers with more memory, and faster computers, are better.

When you purchase Stata, you may install it on any of the above platforms. Stata licenses are not locked to a single operating system.

5.3 Stata/MP, Stata/SE, or Stata/BE

Stata is available in three editions, although perhaps sizes would be a better word. The editions are, from largest to smallest, Stata/MP, Stata/SE, and Stata/BE. (Prior to Stata 17, the various editions of Stata were called flavors, and Stata/BE was called Stata/IC.) If you have a StataNow license, you can take Stata/MP, Stata/SE, and Stata/BE to mean StataNow/MP, StataNow/SE, and StataNow/BE, respectively.

Stata/MP is the multiprocessor version of Stata. It runs on multiple CPUs or on multiple cores, from 2 to 64. Stata/MP uses however many cores you tell it to use (even one), up to the number of cores for which you are licensed. Stata/MP is the fastest version of Stata. Even so, all the details of parallelization are handled internally and you use Stata/MP just like you use any other editions of Stata. You can read about how Stata/MP works and see how its speed increases with more cores at https://www.stata.com/statamp/.

Stata/SE is like Stata/MP, but for single CPUs. Stata/SE will run on multiple CPUs or multiple-core computers, but it will use only one CPU or core. SE stands for standard edition.

In addition to being the fastest version of Stata, Stata/MP is also the largest. Stata/MP allows up to 1,099,511,627,775 observations in theory, but you can undoubtedly run out of memory first. You may have up to 120,000 variables with Stata/MP. Statistical models may have up to 65,532 variables.

Stata/SE allows up to 2,147,583,647 observations, assuming you have enough memory. You may have up to 32,767 variables with Stata/SE. Statistical models may have up to 10,998 variables.

Stata/BE is the basic version of Stata. Up to 2,147,583,647 observations and 2,048 variables are allowed, depending on memory. Statistical models may have up to 800 variables.

5.3.1 Determining which version you own

Check your License and Activation Key. Included with every copy of Stata is a License and Activation Key that contains codes that you will input during installation. This determines which editions of Stata you have and for which platform.

Contact us or your distributor if you want to upgrade from one edition to another. Usually, all you need is an upgraded License and Activation Key with the appropriate codes.

If you purchased one edition of Stata and want to use a lesser version, you may. You might want to do this if you had a large computer at work and a smaller one at home. Please remember, however, that you have only one license (or however many licenses you purchased). You may, both legally and ethically, install Stata on both computers and then use one or the other, but you should not use them both simultaneously.

5.3.2 Determining which version is installed

If Stata is already installed, you can find out which Stata you are using by entering Stata as you normally do and typing `about`:

```
. about

StataNow/MP 19.5 for Windows (64-bit x86-64)
Revision date
Copyright 1985-2025 StataCorp LLC

Total usable memory: 8388608 KB

Stata license: 10-user 32-core network perpetual
Serial number: 19
  Licensed to: Stata Developer
               StataCorp LLC
```

5.4 Size limits of Stata/MP, SE, and BE

Stata/MP allows more variables and observations, longer macros, and a longer command line than Stata/SE. Stata/SE allows more variables, larger models, longer macros, and a longer command line than Stata/BE. The longer command line and macro length are required because of the greater number of variables allowed. The larger model means that Stata/MP and Stata/SE can fit statistical models with more independent variables. See [R] **Limits** for the maximum size limits for Stata/MP, Stata/SE, and Stata/BE.

5.5 Speed comparison of Stata/MP, SE, and BE

We have written a white paper called the Stata/MP Performance Report, which compares the performance of Stata/MP with Stata/SE. The paper is available at https://www.stata.com/statamp/. The white paper includes command-by-command performance measurements.

In summary, on a dual-core computer, Stata/MP will run commands in 71% of the time required by Stata/SE. There is variation; some commands run in half the time and others are not sped up at all. Statistical estimation commands run in 59% of the time. Numbers quoted are medians. Average performance gains are higher because commands that take longer to execute are generally sped up more.

Stata/MP running on four cores runs in 50% (all commands) and 35% (estimation commands) of the time required by Stata/SE. Both numbers are median measures.

Stata/MP supports up to 64 cores.

Stata/BE is slower than Stata/SE, but those differences emerge only when processing datasets that are pushing the limits of Stata/BE. Stata/SE has a larger memory footprint and uses that extra memory for larger look-aside tables to more efficiently process large datasets. The real benefits of the larger tables become apparent only after exceeding the limits of Stata/BE. Stata/SE was designed for processing large datasets.

The differences are all technical and internal. From the user's point of view, Stata/MP, Stata/SE, and Stata/BE work the same way.

5.6 Feature comparison of Stata/MP, SE, and BE

The features of MP, SE, and BE editions of Stata on all platforms are the same. The differences are in speed and in limits as discussed above. To learn more, type `help stata/mp`, `help stata/se`, or `help stata/be`.

The StataNow version of Stata contains additional features as listed at https://www.stata.com/new-in-stata/features/#statanow. These features are the same across the MP, SE, and BE editions of StataNow on all platforms.

6 Managing memory

Contents

6.1 Memory-size considerations

Stata works with a copy of data that it loads into memory. To be precise, Stata can work with multiple datasets in memory at the same time. See [D] **frames intro**.

Memory allocation is automatic. Stata automatically sizes itself up and down as your session progresses. Stata obtains memory from the operating system and draws no distinction between real and virtual memory. Virtual memory is memory that resides on disk that operating systems supply when physical memory runs short. Virtual memory is slow but adequate in cases when you have a dataset that is too large to load into real memory. If you wish to limit the maximum amount of memory Stata can use, you can set `max_memory`; see [D] **memory**. If you use the Linux operating system, we strongly suggest you set `max_memory`; see *Serious bug in Linux OS* in [D] **memory**.

6.2 Compressing data

Stata stores data in memory. The `compress` command reduces the amount of memory required to store the data without loss of precision or any other disadvantages; see [D] **compress**. Typing `compress` every so often is a good idea.

`compress` works by examining the values you have stored and changing the data types of variables when that can be done without loss of precision. For instance, you may have a variable stored as `float` but that records only integer values between −127 and 100. `compress` would change the storage type of that variable to `byte` and save 3 bytes per observation. If you had 100 variables like that, the savings would be 300 bytes per observation, and if you had 3,000,000 observations, the total savings would be nearly 900 megabytes.

6.3 Setting maxvar

If you get the error message “no room to add more variables”, r(900), do not jump to the conclusion that you have exceeded Stata’s capacity.

`maxvar` specifies the maximum number of variables you can use. The default setting depends on whether you are using Stata/MP, Stata/SE, or Stata/BE. To determine the current setting, type `query memory` at the Stata prompt.

If you use Stata/MP, you can reset this maximum number to 120,000. If you use Stata/SE, you can reset this maximum number to 32,767. Set `maxvar` to more than you need—at least 20 more than you need but not too much more than you need. Figure that each 10,000 variables consumes roughly 0.5 megabytes of memory.

You reset maxvar using the `set maxvar` command,

`set maxvar # [ , permanently ]`

where 2,048 $\leq$ # $\leq$ 120,000, depending on your edition of Stata. You can reset `maxvar` repeatedly during a session. If you specify the `permanently` option, you change `maxvar` not only for this session but also for future sessions. Each additional 10,000 variables specified with `set maxvar` requires Stata to set aside roughly 1.3 megabytes of memory for variable names, not including the data stored in those variables.

6.4 The memory command

The `memory` command will show you the major components of Stata's memory footprint.

```
. use https://www.stata-press.com/data/r19/regsmpl
(NLS women 14-26 in 1968)

. memory

Memory usage
                                         Used                Allocated
----------------------------------------------------------------------
Data                                  856,020               67,108,864
strLs                                       0                        0
----------------------------------------------------------------------
Data & strLs                          856,020               67,108,864
----------------------------------------------------------------------
Data & strLs                          856,020               67,108,864
Variable names, %fmts, ...              4,644                  182,379
Overhead                            1,081,344                1,081,744

Stata matrices                              0                        0
ado-files                              34,589                   34,589
Stored results                              0                        0

Mata matrices                               0                        0
Mata functions                              0                        0

set maxvar usage                    5,281,738                5,281,738

Other                                   4,066                    4,066
----------------------------------------------------------------------
Total                               7,252,861               73,693,380
```

See [D] **memory**.

6.5 Setting aside memory for temporary storage of preserved datasets

Stata has a feature to `preserve` and `restore` datasets, allowing you to manipulate the data during an analysis and bring them back without harm. Stata/MP uses memory to make copies of these datasets as fast as possible. Stata/SE and Stata/BE make the copies on disk.

To control the amount of memory Stata/MP will use for these temporary dataset copies before it falls back to slower disk storage, use the `set max_preservemem` setting. See [P] **preserve** for more details.

7 –more– conditions

Contents

7.1 Description

By default, Stata does not pause its output. If a command generates more than a screenful of output, you can scroll back to see what you missed.

Some users prefer for Stata to pause every time the screen is full of output. You can enable this with Stata's `set more` command. See [R] **more**.

If you `set more on`, Stata will pause any time a command generates more than a screenful of output. When you see —`more`— at the bottom of the screen,

Press ...	and Stata ...
letter *l* or *Enter*	displays the next line
letter *q*	acts as if you pressed *Break*
Spacebar or any other key	displays the next screen

Also, from the menu, you can press the *More* button, the green button with the down arrow.

—`more`— is Stata's way of telling you that it has something more to show you, but showing you that something more will cause the information on the screen to scroll off.

7.2 The set more command

If you type `set more on`, —`more`— conditions will arise at the appropriate places.

If you type `set more off` (Stata's default behavior), —`more`— conditions will never arise and Stata's output will scroll by at full speed.

Programmers: If `set more` is used within a do-file or program, Stata automatically restores the previous `set more` setting when the do-file or program concludes.

See [R] **more**.

7.3 The more programming command

Ado-file programmers need take no special action to have —`more`— conditions arise when the screen is full. Stata handles that automatically.

If, however, you wish to force a —`more`— condition early, you can include the `more` command in your program. Simply type `more`, because the command takes no arguments.

For more information, see [P] **more**.

8 Error messages and return codes

Contents

8.1 Making mistakes

When an error occurs, Stata produces an error message and a *return code*. For instance,

```
. list myvar
no variables defined
r(111);
```

We ask Stata to list the variable named `myvar`. Because we have no data in memory, Stata responds with the message “no variables defined” and a line that reads “`r(111)`”.

The “no variables defined” is called the error message.

The 111 is called the return code. You can click on blue return codes to get a detailed explanation of the error.

8.1.1 Mistakes are forgiven

After “no variables defined” and `r(111)`, all is forgiven; it is as if the error never occurred.

Typically, the message will be enough to guide you to a solution, but if it is not, the numeric return codes are documented in [P] **error**.

8.1.2 Mistakes stop user-written programs and do-files

Whenever an error occurs in a user-written program or do-file, the program or do-file immediately stops execution and the error message and return code are displayed.

For instance, consider the following do-file:

begin `myfile.do`

```
use https://www.stata-press.com/data/r19/auto
decribe
list
```

end `myfile.do`

Note the second line—you meant to type describe but typed decribe. Here is what happens when you execute this do-file by typing do myfile:

```
. do myfile

. use https://www.stata-press.com/data/r19/auto
(1978 automobile data)

. decribe
command decribe is unrecognized
r(199);

end of do-file
r(199);

. _
```

The first error message and return code were caused by the illegal decribe. This then caused the do-file itself to be aborted; the valid list command was never executed.

8.1.3 Advanced programming to tolerate errors

Errors are not only of the typographical kind; some are substantive. A command that is valid in one dataset might not be valid in another. Moreover, in advanced programming, errors are sometimes anticipated: use one dataset if it is there, but use another if you must.

Programmers can access the return code to determine whether an error occurred, which they can then ignore, or, by examining the return code, code their programs to take the appropriate action. This is discussed in [P] **capture**.

You can also prevent do-files from stopping when errors occur by using the do command's nostop option.

```
. do myfile, nostop
```

8.2 The return message for obtaining command timings

In addition to error messages and return codes, there is something called a return message, which you normally do not see. Normally, if you typed summarize tempjan, you would see

```
. use https://www.stata-press.com/data/r19/citytemp
(City temperature data)

. summarize tempjan

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
     tempjan |        954    35.74895    14.18813        2.2       72.6
```

If you were to type

```
. set rmsg on
r; t=0.00 10:21:22
```

sometime during your session, Stata would display return messages:

```
. summarize tempjan
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
     tempjan |        954    35.74895    14.18813        2.2       72.6
r; t=0.01 10:21:26
```

The line that reads `r; t=0.01 10:21:26` is called the return message.

The `r;` indicates that Stata successfully completed the command.

The `t=0.01` shows the amount of time, in seconds, it took Stata to perform the command (timed from the point you pressed *Enter* to the time Stata typed the message). This command took a hundredth of a second. Stata also shows the time of day with a 24-hour clock. This command completed at 10:21 a.m.

Stata can run commands stored in files (called do-files) and can log output. Some users find the detailed return message helpful with do-files. They construct a long program and let it run overnight, logging the output. They come back the next morning, look at the output, and discover a mistake in some portion of the job. They can look at the return messages to determine how long it will take to rerun that portion of the program.

You may `set rmsg on` whenever you wish.

When you want Stata to stop displaying the detailed return message, type `set rmsg off`.

There is another way to obtain timings of subsets of code that is of interest to programmers. See [P] **timer**.

9 The Break key

Contents

9.1 Making Stata stop what it is doing

When you want to make Stata stop what it is doing and return to the Stata dot prompt, you click on *Break*:

Stata for Windows:	click on the **Break** button (it is the button with the big red X), or press *Ctrl+Pause/Break*
Stata for Mac:	click on the **Break** button or press *Command+.* (period)
Stata for Unix(GUI):	click on the **Break** button or press *Ctrl+k*
Stata for Unix(console):	press *Ctrl+c* or press *q*

Elsewhere in this manual, we describe this action as simply clicking on *Break*. Break tells Stata to cancel what it is doing and return control to you as soon as possible.

If you click on *Break* in response to the input prompt or while you are typing a line, Stata ignores it, because you are already in control.

If you click on *Break* while Stata is doing something—creating a new variable, sorting a dataset, making a graph, etc.—Stata stops what it is doing, undoes it, and issues an input prompt. The state of the system is the same as if you had never issued the command.

▷ Example 1

You are fitting a logit model, type the command, and, as Stata is working on the problem, realize that you omitted an important variable:

```
. logit foreign mpg weight
Iteration 0:  Log likelihood =  -45.03321
Iteration 1:  Log likelihood = -29.898968
--Break--
r(1);

. _
```

When you clicked on *Break*, Stata responded by displaying —`Break`— and `r(1);`. Clicking on *Break* always results in a return code of 1—that is why return codes are called return codes and not error codes. The 1 does not indicate an error, but it does indicate that the command did not complete its task.

◁

9.2 Side effects of clicking on Break

In general, there are no side effects of clicking on Break. We said above that Stata undoes what it is doing so that the state of the system is the same as if you had never issued the command. There are two exceptions to that statement.

If you are reading data from disk by using `import delimited`, `infile`, or `infix`, whatever data have already been read will be left behind in memory, the theory being that perhaps you stopped the process so you could verify that you were reading the right data correctly before sitting through the whole process. If not, you can always `clear`.

```
. infile v1-v9 using workdata
(eof not at end of obs)
(4 observations read)
—Break—
r(1);
```

The other exception is `sort`. You have a large dataset in memory, decide to sort it, and then change your mind.

```
. sort price
—Break—
r(1);
```

If the dataset was previously sorted by, say, the variable `prodid`, it is no longer. When you click on *Break* in the middle of a `sort`, Stata marks the data as unsorted.

9.3 Programming considerations

There are basically no programming considerations for handling Break because Stata handles it all automatically. If you write a program or do-file, execute it, and then click on *Break*, Stata stops execution just as it would with an internal command.

Advanced programmers may be concerned about cleaning up after themselves; perhaps they have generated a temporary variable they intended to drop later or a temporary file they intended to erase later. If a Stata user clicks on *Break*, how can you ensure that these temporary variables and files will be erased?

If you obtain names for such temporary items from Stata's `tempname`, `tempvar`, and `tempfile` commands, Stata will automatically erase the temporary items; see [U] **18.7 Temporary objects**.

There are instances, however, when a program must commit to executing a group of commands without interruption, or the user's data would be left in an intermediate or undefined state. In these instances, Stata provides a

```
nobreak {
                ...
        }
```

construct; see [P] **break**. Also see [M-5] **setbreakintr()** to read about Break-key processing in Mata.

10 Keyboard use

Contents

10.1 Description

The keyboard should operate much the way you would expect, with a few additions:

- There are some unexpected keys you can press to obtain previous commands you have typed. Also, you can click once on a command in the History window to reload it, or click on it twice to reload and execute; this feature is discussed in the *Getting Started* manuals.
- There are a host of command-editing features for Stata for Unix(console) users because their user interface does not offer such features.
- Regardless of operating system or user interface, if there are *F*-keys on your keyboard, they have special meaning and you can change the definitions of the keys.

10.2 F-keys

Windows users: *F3* and *F10* are reserved internally by Windows; you cannot program these keys.

By default, Stata defines the *F*-keys to mean

F-key	Definition
F1	`help advice;`
F2	`describe;`
F7	`save`
F8	`use`

The semicolons at the end of some entries indicate an implied *Enter*.

Stata provides several methods for obtaining help. To learn about these methods, select **Help > Advice**. Or you can just press *F1*.

`describe` is the Stata command to report the contents of data loaded into memory. It is explained in [D] **describe**. Normally, you type `describe` and press *Enter*. You can also press *F2*.

`save` is the command to save the data in memory into a file, and `use` is the command to load data; see [D] **use** and [D] **save**. The syntax of each is the same: `save` or `use` followed by a filename. You can type the commands or you can press *F7* or *F8* followed by the filename.

You can change the definitions of the *F*-keys. For instance, the command to list data is `list`; you can read about it in [D] **list**. The syntax is `list` to list all the data, or `list` followed by the names of some variables to list just those variables (there are other possibilities).

If you wanted *F9* to mean `list`, you could type

```
. global F9 "list "
```

In the above, `F9` refers to the letter *F* followed by *9*, not the *F9* key. Note the capitalization and spacing of the command.

You type `global` in lowercase, type `F9`, and then type `"list "`. The space at the end of `list` is important. In the future, rather than typing `list mpg weight`, you want to be able to press the *F9* key and then type only `mpg weight`. You put a space in the definition of `F9` so that you would not have to type a space in front of the first variable name after pressing *F9*.

Now say you wanted *F5* to mean list all the data—`list` followed by *Enter*. You could define

```
. global F5 "list;"
```

Now you would have two ways of listing all the data: press *F9*, and then press *Enter*, or press *F5*. The semicolon at the end of the definition of *F5* will press *Enter* for you.

If you really want to change the definitions of *F9* and *F5*, you will probably want to change the definition every time you invoke Stata. One way would be to type the two `global` commands every time you invoke Stata. Another way would be to type the two commands into a text file named `profile.do`. Stata executes the commands in `profile.do` every time it is launched if `profile.do` is placed in the appropriate directory:

Windows:	see [GSW] **B.3 Executing commands every time Stata is started**
Mac:	see [GSM] **B.1 Executing commands every time Stata is started**
Unix:	see [GSU] **B.1 Executing commands every time Stata is started**

You can use the *F*-keys any way you desire: they contain a string of characters, and pressing the *F*-key is equivalent to typing those characters.

❑ Technical note

[*Stata for Unix(console) users.*] Sometimes Unix assigns a special meaning to the *F*-keys, and if it does, those meanings supersede our meanings. Stata provides a second way to get to the *F*-keys. Press *Ctrl+F*, release the keys, and then press a number from `0` through `9`. Stata interprets *Ctrl+F* plus 1 as equivalent to the *F1* key, *Ctrl+F* plus 2 as *F2*, and so on. *Ctrl+F* plus 0 means *F10*. These keys will work only if they are properly mapped in your `termcap` or `terminfo` entry.

❑

❑ Technical note

On some international keyboards, the left single quote is used as an accent character. In this case, we recommend mapping this character to one of your function keys. In fact, you might find it convenient to map both the left single quote (`‘`) and the right single quote (`’`) characters so that they are next to each other.

Within Stata, open the Do-file Editor. Type the following two lines in the Do-file Editor:

```
global F4 ‘
global F5 ’
```

Save the file as `profile.do` into your Stata directory. If you already have a `profile.do` file, append the two lines to your existing `profile.do` file.

Exit Stata and restart it. You should see the startup message

```
running C:\Program Files\Stata19\profile.do ...
```

or some variant of it depending on where your Stata is installed. Press *F4* and *F5* to verify that they work.

If you did not see the startup message, you did not save the `profile.do` in your home folder.

You can, of course, map to any other function keys, but *F1*, *F2*, *F7*, and *F8* are already used.

❑

10.3 Editing keys in Stata

Users have available to them the standard editing keys for their operating system. So, Stata should just edit what you type in the natural way—the Stata Command window is a standard edit window.

Also, you can fetch commands from the History window into the Command window. Click on a command in the History window, and it is loaded into the Command window, where you can edit it. Alternatively, if you double-click on a line in the History window, it is loaded and executed.

Another way to get lines from the History window into the Command window is with the *PgUp* and *PgDn* keys. Press *PgUp* and Stata loads the last command you typed into the Command window. Press it again and Stata loads the line before that, and so on. *PgDn* goes in the opposite direction.

Another editing key that interests users is *Esc*. This key clears the Command window.

In summary,

Press	Result
PgUp	Steps back through commands and moves command from History window to Command window
PgDn	Steps forward through commands and moves command from History window to Command window
Esc	Clears Command window

10.4 Editing keys in Stata for Unix(console)

Certain keys allow you to edit the line that you are typing. Because Stata supports a variety of computers and keyboards, the location and the names of the editing keys are not the same for all Stata users.

Every keyboard has the standard alphabet keys (*QWERTY* and so on), and every keyboard has a *Ctrl* key. Some keyboards have extra keys located to the right, above, or left, with names like *PgUp* and *PgDn*.

Throughout this manual we will refer to Stata's editing keys using names that appear on nobody's keyboard. For instance, PrevLine is one of the Stata editing keys—it retrieves a previous line. Hunt all you want, but you will not find it on your keyboard. So, where is PrevLine? We have tried to put it where you would naturally expect it. On keyboards with a key labeled *PgUp*, *PgUp* is the PrevLine key, but on everybody's keyboard, no matter which version of Unix, brand of keyboard, or anything else, *Ctrl+R* also means PrevLine.

When we say press PrevLine, now you know what we mean: press *PgUp* or *Ctrl+R*. The editing keys are the following:

Name for editing key	Editing key	Function
Kill	*Esc* on PCs and *Ctrl+U*	Deletes the line and lets you start over.
Dbs	*Backspace* on PCs and *Backspace* or *Delete* on other computers	Backs up and deletes one character.
Lft	←, *4* on the numeric keypad for PCs, and *Ctrl+H*	Moves the cursor left one character without deleting any characters.
Rgt	→, *6* on the numeric keypad for PCs, and *Ctrl+L*	Moves the cursor forward one character.
Up	↑, *8* on the numeric keypad for PCs, and *Ctrl+O*	Moves the cursor up one physical line on a line that takes more than one physical line. Also see PrevLine.
Dn	↓, *2* on the numeric keypad for PCs, and *Ctrl+N*	Moves the cursor down one physical line on a line that takes more than one physical line. Also see NextLine.
PrevLine	*PgUp* and *Ctrl+R*	Retrieves a previously typed line. You may press PrevLine multiple times to step back through previous commands.
NextLine	*PgDn* and *Ctrl+B*	The inverse of PrevLine.
Seek	*Ctrl+Home* on PCs and *Ctrl+W*	Goes to the line number specified. Before pressing Seek, type the line number. For instance, typing *3* and then pressing Seek is the same as pressing PrevLine three times.
Ins	*Ins* and *Ctrl+E*	Toggles insert mode. In insert mode, characters typed are inserted at the position of the cursor.
Del	*Del* and *Ctrl+D*	Deletes the character at the position of the cursor.
Home	*Home* and *Ctrl+K*	Moves the cursor to the start of the line.
End	*End* and *Ctrl+P*	Moves the cursor to the end of the line.
Hack	*Ctrl+End* on PCs, and *Ctrl+X*	Hacks off the line at the cursor.
Tab	→| on PCs, *Tab*, and *Ctrl+I*	Expand variable name.
Btab	|← on PCs, and *Ctrl+G*	The inverse of Tab.

▷ Example 1

It is difficult to demonstrate the use of editing keys in print. You should try each of them. Nevertheless, here is an example:

```
. summarize price waht
```

You typed `summarize price waht` and then pressed the *Lft* key (← key or *Ctrl+H*) three times to maneuver the cursor back to the `a` of `waht`. If you were to press *Enter* right now, Stata would see the command `summarize price waht`, so where the cursor is does not matter when you press *Enter*. If you wanted to execute the command `summarize price`, you could back up one more character and then press the Hack key. We will assume, however, that you meant to type `weight`.

If you were now to press the letter *e* on the keyboard, an `e` would appear on the screen to replace the `a`, and the cursor would move under the character `h`. We now have `weht`. You press *Ins*, putting Stata into insert mode, and press *i* and *g*. The line now says `summarize price weight`, which is correct, so you press *Enter*. We did not have to press *Ins* before every character we wanted to insert. The *Ins* key is a toggle: If we press it again, Stata turns off insert mode, and what we type replaces what was there. When we press *Enter*, Stata forgets all about insert mode, so we do not have to remember from one command to the next whether we are in insert mode.

◁

❑ Technical note

Stata performs its editing magic from the information about your terminal recorded in `/etc/termcap`(5) or, under System V, `/usr/lib/terminfo`(4). If some feature does not appear to work, the entry for your terminal in the `termcap` file or `terminfo` directory is probably incorrect. Contact your system administrator.

❑

10.5 Editing previous lines in Stata

In addition to what is said below, remember that the History window also shows the contents of the review buffer.

One way to retrieve lines is with the PrevLine and NextLine keys. Remember, PrevLine and NextLine are the names we attach to these keys—there are no such keys on your keyboard. You have to look back at the previous section to find out which keys correspond to PrevLine and NextLine on your computer. To save you the effort this time, PrevLine probably corresponds to *PgUp* and NextLine probably corresponds to *PgDn*.

Suppose you wanted to reissue the third line back. You could press PrevLine three times and then press *Enter*. If you made a mistake and pressed PrevLine four times, you could press NextLine to go forward in the buffer. You do not have to count lines because, each time you press PrevLine or NextLine, the current line is displayed on your monitor. Simply press the key until you find the line you want.

Another method for reviewing previous lines, `#review`, is convenient for Unix(console) users.

▷ Example 2

Typing `#review` by itself causes Stata to list the last five commands you typed. For instance,

```
. #review
5 list make mpg weight if abs(res)>6
4 list make mpg weight if abs(res)>5
3 tabulate foreign if abs(res)>5
2 regress mpg weight weight2
1 test weight2=0
. _
```

We can see from the listing that the last command typed by the user was `test weight2=0`. Or, you may just look at the History window to see the history of commands you typed.

◁

▷ Example 3

Perhaps the command you are looking for is not among the last five commands you typed. You can tell Stata to go back any number of lines. For instance, typing `#review 15` tells Stata to show you the last 15 lines you typed:

```
. #review 15
%20 regress mpg weight weight2
%19 generate predmpg=_pred
%18 generate resmpg=mpg-_pred
%17 summarize resmpg, detail
%16 regress mpg weight weight2 foreign
15 replace resmpg=mpg-pred
14 summarize resmpg, detail
13 drop predmpg
12 describe
11 sort foreign
10 by foreign: summarize mpg weight
9 * lines that start with a * are comments.
8 * they go into the review buffer too.
7 summarize resmpg, detail
6 list make mpg weight
5 list make mpg weight if abs(res)>6
4 list make mpg weight if abs(res)>5
3 tabulate foreign if abs(res)>5
2 regress mpg weight weight2
1 test weight2=0
. _
```

If you wanted to resubmit the 10th previous line, you could type `10` and press Seek, or you could press PrevLine 10 times. No matter which of the above methods you prefer for retrieving lines, you may edit previous lines by using the editing keys.

◁

10.6 Tab expansion of variable names

Another way to quickly enter a variable name is to take advantage of Stata's tab-completion feature. Simply type the first few letters of the variable name in the Command window and press the *Tab* key. Stata will automatically type the rest of the variable name for you. If more than one variable name matches the letters you have typed, Stata will complete as much as it can and beep at you to let you know that you have typed a nonunique variable abbreviation.

The tab-completion feature also applies to typing filenames. If you start by typing a double quote, `"`, you can type the first few letters of a filename or directory and press the *Tab* key. Stata will automatically type the rest of the name for you. If more than one filename or directory matches the letters you have typed, Stata will complete as much as it can and beep at you to let you know that you have typed a nonunique abbreviation. After the entire filename or directory has been typed, type another double quote.

Elements of Stata

11 Language syntax

Contents

11.1 Overview

With few exceptions, the basic Stata language syntax is

[by *varlist*:] *command* [*varlist*] [=*exp*] [if *exp*] [in *range*] [*weight*] [, *options*] where square brackets distinguish optional qualifiers and options from required ones. 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, *weight* denotes a weighting expression, and *options* denotes a list of options.

11.1.1 varlist

Most commands that take a subsequent *varlist* do not require that you explicitly type one. If no *varlist* appears, these commands assume *varlist* of `_all`, the Stata shorthand for indicating all the variables in the dataset. In commands that alter or destroy data, Stata requires that the *varlist* be specified explicitly. See [U] **11.4 varname and varlists** for a complete description.

Some commands take *varname*, rather than *varlist*. *varname* refers to exactly one variable. The `tabulate` command requires *varname*; see [R] **tabulate oneway**.

▷ Example 1

The `summarize` command lists the mean, standard deviation, and range of the specified variables. In [R] **summarize**, we see that the syntax diagram for summarize is

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

Farther down on the manual page is a table summarizing `options`, but let's focus on the syntax diagram itself first. Because everything except the word `summarize` is enclosed in square brackets, the simplest form of the command is "`summarize`". Typing `summarize` without arguments is equivalent to typing `summarize _all`; all the variables in the dataset are summarized. Underlining denotes the shortest allowed abbreviation, so we could have typed just `su`; see [U] **11.2 Abbreviation rules**.

The table that defines *options* looks like this:

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

Thus we learn we could also type, for instance, `summarize, detail` or `summarize, detail format`.

As another example, the `drop` command eliminates variables or observations from a dataset. When dropping variables, its syntax is

`drop` *varlist*

`drop` has no option table because it has no options.

In fact, nothing is optional. Typing `drop` by itself would result in the error message "varlist or in range required". To drop all the variables in the dataset, we must type `drop _all`.

Even before looking at the syntax diagram, we could have predicted that *varlist* would be required—`drop` is destructive, so Stata requires us to spell out our intent. The syntax diagram informs us that *varlist* is required because *varlist* is not enclosed in square brackets. Because `drop` is not underlined, it cannot be abbreviated.

◁

11.1.2 by varlist:

The by *varlist*: prefix causes Stata to repeat a command for each subset of the data for which the values of the variables in *varlist* are equal. When prefixed with by *varlist*:, the result of the command will be the same as if you had formed separate datasets for each group of observations, saved them, and then gave the command on each dataset separately. The data must already be sorted by *varlist*, although by has a sort option; see **[U] 11.5 by varlist: construct** for more information.

▷ Example 2

Typing summarize marriage_rate divorce_rate produces a table of the mean, standard deviation, and range of marriage_rate and divorce_rate, using all the observations in the data:

```
. use https://www.stata-press.com/data/r19/census12
(1980 Census data by state)
. summarize marriage_rate divorce_rate
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |         50    .0133221    .0188122   .0074654   .1428282
divorce_rate |         50    .0056641    .0022473   .0029436   .0172918
```

Typing by region: summarize marriage_rate divorce_rate produces one table for each region of the country:

```
. sort region
. by region: summarize marriage_rate divorce_rate
--------------------------------------------------------------------------------
-> region = N Cntrl
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |         12    .0099121    .0011326   .0087363   .0127394
divorce_rate |         12    .0046974    .0011315   .0032817   .0072868
--------------------------------------------------------------------------------
-> region = NE
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |          9    .0087811     .001191   .0075757   .0107055
divorce_rate |          9     .004207    .0010264   .0029436   .0057071
--------------------------------------------------------------------------------
-> region = South
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |         16    .0114654    .0025721   .0074654   .0172704
divorce_rate |         16     .005633    .0013355   .0038917   .0080078
--------------------------------------------------------------------------------
-> region = West
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |         13    .0218987    .0363775   .0087365   .1428282
divorce_rate |         13    .0076037    .0031486   .0046004   .0172918
```

The dataset must be sorted on the by variables:

```
. use https://www.stata-press.com/data/r19/census12
(1980 Census data by state)

. by region: summarize marriage_rate divorce_rate
not sorted
r(5);

. sort region

. by region: summarize marriage_rate divorce_rate
(output appears)
```

We could also have asked that by sort the data:

```
. by region, sort: summarize marriage_rate divorce_rate
(output appears)
```

`by` *varlist*`:` can be used with most Stata commands; we can tell which ones by looking at their syntax diagrams. For instance, we could obtain the correlations by `region`, between `marriage_rate` and `divorce_rate`, by typing `by region: correlate marriage_rate divorce_rate`.

◁

❑ Technical note

varlist in `by` *varlist*`:` may contain up to 120,000 variables with Stata/MP, 32,767 variables with Stata/SE, or 2,048 variables with Stata/BE; these are the maximum allowed in the dataset. For instance, if we had data on automobiles and wished to obtain means according to market category (`market`) broken down by manufacturer (`origin`), we could type `by market origin: summarize`. That *varlist* contains two variables: `market` and `origin`. If the data were not already sorted on `market` and `origin`, we would first type `sort market origin`.

varlist in `by` *varlist*`:` may also contain string variables, numeric variables, or both. In the example above, `region` is a string variable, in particular, a `str7`. The example would have worked, however, if `region` were a numeric variable with values 1, 2, 3, and 4, or even 12.2, 16.78, 32.417, and 152.13.

❑

11.1.3 if exp

The `if` *exp* qualifier restricts the scope of a command to those observations for which the value of the expression is *true* (which is equivalent to the expression being nonzero; see **[U] 13 Functions and expressions**).

▷ Example 3

Typing `summarize marriage_rate divorce_rate if region=="West"` produces a table for the western region of the country:

```
. summarize marriage_rate divorce_rate if region == "West"

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |         13    .0218987    .0363775   .0087365   .1428282
divorce_rate |         13    .0076037    .0031486   .0046004   .0172918
```

The double equal sign in `region=="West"` is not an error. Stata uses a *double* equal sign to denote equality testing and one equal sign to denote assignment; see **[U] 13 Functions and expressions**.

A command may have at most one `if` qualifier. If you want the summary for the West restricted to observations with values of `marriage_rate` in excess of 0.015, do *not* type `summarize marriage_rate divorce_rate if region=="West" if marriage_rate>.015`. Instead type

```
. summarize marriage_rate divorce_rate if region == "West" & marriage_rate > .015

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |          1    .1428282           .   .1428282   .1428282
divorce_rate |          1    .0172918           .   .0172918   .0172918
```

You may not use the word *and* in place of the symbol "&" to join conditions. To select observations that meet one condition *or* another, use the "|" symbol. For instance, `summarize marriage_rate divorce_rate if region=="West" | marriage_rate>.015` summarizes all observations for which `region` is West *or* `marriage_rate` is greater than 0.015.

◁

▷ Example 4

`if` may be combined with `by`. Typing `by region: summarize marriage_rate divorce_rate if marriage_rate>.015` produces a set of tables, one for each region, reflecting summary statistics on `marriage_rate` and `divorce_rate` among observations for which `marriage_rate` exceeds 0.015:

```
. by region: summarize marriage_rate divorce_rate if marriage_rate > .015

-------------------------------------------------------------------------------
-> region = N Cntrl

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |          0
divorce_rate |          0

-------------------------------------------------------------------------------
-> region = NE

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |          0
divorce_rate |          0

-------------------------------------------------------------------------------
-> region = South

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |          2    .0163219    .0013414   .0153734   .0172704
divorce_rate |          2    .0061813    .0025831   .0043548   .0080078

-------------------------------------------------------------------------------
-> region = West

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |          1    .1428282           .   .1428282   .1428282
divorce_rate |          1    .0172918           .   .0172918   .0172918
```

The results indicate that there are no states in the Northeast and North Central regions for which `marriage_rate` exceeds 0.015, whereas there are two such states in the South and one state in the West.

◁

11.1.4 in range

The in *range* qualifier restricts the scope of the command to a specific observation range. A range specification takes the form $\#_1$ [/$\#_2$], where $\#_1$ and $\#_2$ are positive or negative integers. Negative integers are understood to mean "from the end of the data", with −1 referring to the last observation. The implied first observation must be less than or equal to the implied last observation.

The first and last observations in the dataset may be denoted by f and l (lowercase letter), respectively. F is allowed as a synonym for f, and L is allowed as a synonym for l. A range specifies absolute observation numbers within a dataset. As a result, the in qualifier may not be used when the command is preceded by the by *varlist*: prefix; see **[U] 11.5 by varlist: construct**.

▷ Example 5

Typing summarize marriage_rate divorce_rate in 5/25 produces a table based on the values of marriage_rate and divorce_rate in observations 5–25:

```
. summarize marriage_rate divorce_rate in 5/25
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |         21    .0093941    .0012851   .0075757     .01293
divorce_rate |         21    .0045305    .0011273   .0029436   .0072868
```

This is, admittedly, a rather odd thing to want to do. It would not be odd, however, if we substituted list for summarize. If we wanted to see the states with the 10 lowest values of marriage_rate, we could type sort marriage_rate followed by list marriage_rate in 1/10.

Typing summarize marriage_rate divorce_rate in f/l is equivalent to typing summarize marriage_rate divorce_rate—all observations are summarized.

◁

▷ Example 6

Typing summarize marriage_rate divorce_rate in 5/25 if region == "South" produces a table based on the values of the two variables in observations 5–25 for which the value of region is South:

```
. summarize marriage_rate divorce_rate in 5/25 if region == "South"
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |          4    .0108187    .0016621   .0089399     .01293
divorce_rate |          4    .0051821    .0009356   .0043054   .0063596
```

The ordering of the in and if qualifiers is not significant. The command could also have been specified as summarize marriage_rate divorce_rate if region == "South" in 5/25.

◁

▷ Example 7

Negative in ranges can be useful with sort. For instance, we have data on automobiles and wish to list the five with the highest mileage ratings:

```
. use https://www.stata-press.com/data/r19/auto
(1978 automobile data)

. sort mpg

. list make mpg in -5/l

     ┌──────────────────────┐
     │ make             mpg │
     ├──────────────────────┤
 70. │ Toyota Corolla    31 │
 71. │ Plym. Champ       34 │
 72. │ Subaru            35 │
 73. │ Datsun 210        35 │
 74. │ VW Diesel         41 │
     └──────────────────────┘
```

◁

11.1.5 =exp

= *exp* specifies the value to be assigned to a variable and is most often used with generate and replace. See [U] **13 Functions and expressions** for details on expressions and [D] **generate** for details on the generate and replace commands.

Expression	Meaning
generate newvar=oldvar+2	creates a new variable named newvar equal to oldvar+2
replace oldvar=oldvar+2	changes the contents of the existing variable oldvar
egen newvar=rank(oldvar)	creates newvar containing the ranks of oldvar (see [D] **egen**)

11.1.6 weight

weight indicates the weight to be attached to each observation. The syntax of *weight* is

[*weightword*=*exp*]

where you actually type the square brackets and where *weightword* is one of

weightword	Meaning
weight	default treatment of weights
fweight *or* frequency	frequency weights
pweight	sampling weights
aweight *or* cellsize	analytic weights
iweight	importance weights

The underlining indicates the minimum acceptable abbreviation. Thus weight may be abbreviated w or we, etc.

▷ Example 8

Before explaining what the different types of weights mean, let's obtain the population-weighted mean of a variable called `median_age` from data containing observations on all 50 states of the United States. The dataset also contains a variable named `pop`, which is the total population of each state.

```
. use https://www.stata-press.com/data/r19/census12
(1980 Census data by state)

. summarize median_age [weight=pop]
(analytic weights assumed)

    Variable |     Obs      Weight        Mean   Std. dev.       Min        Max
-------------+-----------------------------------------------------------------
  median_age |      50   225907472    30.11047    1.66933       24.2       34.7
```

In addition to telling us that our dataset contains 50 observations, Stata informs us that the sum of the weight is 225,907,472, which was the number of people living in the United States as of the 1980 census. The weighted mean is 30.11. We were also informed that Stata assumed that we wanted "analytic" weights.

◁

`weight` is each command's idea of what the "natural" weights are and is one of `fweight`, `pweight`, `aweight`, or `iweight`. When you specify the vague `weight`, the command informs you which kind it assumes. Not every command supports every kind of weight. A note below the syntax diagram for a command will tell you which weights the command supports.

Stata understands four kinds of weights:

1. `fweight`s, or frequency weights, indicate duplicated observations. `fweight`s are always integers. If the `fweight` associated with an observation is 5, that means there are really 5 such observations, each identical.

2. `pweight`s, or sampling weights, denote the inverse of the probability that this observation is included in the sample because of the sampling design. A `pweight` of 100, for instance, indicates that this observation is representative of 100 subjects in the underlying population. The scale of these weights does not matter in terms of estimated parameters and standard errors, except when estimating totals and computing finite-population corrections with the `svy` commands; see [SVY] **Survey**.

3. `aweight`s, or analytic weights, are inversely proportional to the variance of an observation; that is, the variance of the jth observation is assumed to be σ^2/w_j, where w_j are the weights. Typically, the observations represent averages, and the weights are the number of elements that gave rise to the average. For most Stata commands, the recorded scale of `aweight`s is irrelevant; Stata internally rescales them to sum to N, the number of observations in your data, when it uses them.

4. `iweight`s, or importance weights, indicate the relative "importance" of the observation. They have no formal statistical definition; this is a catch-all category. Any command that supports `iweight`s will define how they are treated. They are usually intended for use by programmers who want to produce a certain computation.

See [U] **20.24 Weighted estimation** for a thorough discussion of weights and their meaning.

❑ Technical note

When you do not specify a weight, the result is equivalent to specifying `[fweight=1]`.

❑

11.1.7 options

Many commands take command-specific options. These are described along with each command in the *Reference* manuals. Options are indicated by typing a comma at the end of the command, followed by the options you want to use.

▷ Example 9

Typing `summarize marriage_rate` produces a table of the mean, standard deviation, minimum, and maximum of the variable `marriage_rate`:

```
. summarize marriage_rate
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
marriage_r~e |         50    .0133221    .0188122   .0074654   .1428282
```

The syntax diagram for `summarize` is

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

followed by the option table

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

Thus the options allowed by `summarize` are `detail` or `meanonly`, `format`, and `separator()`. The shortest allowed abbreviations for these options are `d` for `detail`, `mean` for `meanonly`, `f` for `format`, and `sep()` for `separator()`; see **[U] 11.2 Abbreviation rules**.

Typing `summarize marriage_rate, detail` produces a table that also includes selected percentiles, the four largest and four smallest values, the skewness, and the kurtosis.

```
. summarize marriage_rate, detail
                        marriage_rate
-------------------------------------------------------------
      Percentiles      Smallest
 1%     .0074654       .0074654
 5%     .0078956       .0075757
10%     .0080043       .0078956       Obs                  50
25%     .0089399       .0079079       Sum of wgt.          50

50%     .0105669                      Mean           .0133221
                        Largest       Std. dev.      .0188122
75%     .0122899       .0146266
90%     .0137832       .0153734       Variance       .0003539
95%     .0153734       .0172704       Skewness       6.718494
99%     .1428282       .1428282       Kurtosis       46.77306
```

◁

Some commands have options that are required. For instance, the `ranksum` command requires the `by(`*groupvar*`)` option, which identifies the grouping variable. *groupvar* is a specific kind of *varname*. It identifies to which group each observation belongs.

❑ Technical note

Once you have typed the *varlist* for the command, you can place options anywhere in the command. You can type `summarize marriage_rate divorce_rate if region=="West", detail`, or you can type `summarize marriage_rate divorce_rate, detail, if region=="West"`. You use a second comma to indicate a return to the command line as opposed to the option list. Leaving out the comma after the word `detail` would cause an error because Stata would attempt to interpret the phrase `if region=="West"` as an option rather than as part of the command.

You may not type an option in the middle of a *varlist*. Typing `summarize marriage_rate, detail, divorce_rate` will result in an error.

Options need not be specified contiguously. You may type `summarize marriage_rate divorce_rate, detail, if region=="South", noformat`. Both `detail` and `noformat` are options.

❑

❑ Technical note

Most options are toggles—they indicate that something either is or is not to be done. Sometimes it is difficult to remember which is the default. The following rule applies to all options: if *option* is an option, then `no`*option* is an option as well, and vice versa. Thus if we could not remember whether `detail` or `nodetail` were the default for `summarize` but we knew that we did not want the detail, we could type `summarize, nodetail`. Typing the `nodetail` option is unnecessary, but Stata will not complain.

Some options take *arguments*. The Stata `kdensity` command has an `n(`#`)` option that indicates the number of points at which the density estimate is to be evaluated. When an option takes an argument, the argument is enclosed in parentheses.

Some options take more than one argument. In such cases, arguments should be separated from one another by commas. For instance, you might see in a syntax diagram

`saving(`*filename*[`, replace`]`)`

Here `replace` is the (optional) second argument. *Lists*, such as lists of variables (varlists) and lists of numbers (numlists), are considered to be one argument. If a syntax diagram reported

`powers(`*numlist*`)`

the list of numbers would be one argument, so the elements would not be separated by commas. You would type, for instance, `powers(1 2 3 4)`. In fact, Stata will tolerate commas here, so you could type `powers(1,2,3,4)`.

Some options take string arguments. `regress` has an `eform()` option that works this way—for instance, `eform("Exp Beta")`. To play it safe, you should type the quotes surrounding the string, although it is not required. If you do not type the quotes, any sequence of two or more consecutive blanks will be interpreted as one blank. Thus `eform(Exp  beta)` would be interpreted the same as `eform(Exp beta)`.

❑

11.1.8 numlist

A *numlist* is a list of numbers. Stata allows certain shorthands to indicate ranges:

Numlist	Meaning
`2`	just one number
`1 2 3`	three numbers
`3 2 1`	three numbers in reversed order
`.5 1 1.5`	three different numbers
`1 3 -2.17 5.12`	four numbers in jumbled order
`1/3`	three numbers: 1, 2, 3
`3/1`	the same three numbers in reverse order
`5/8`	four numbers: 5, 6, 7, 8
`-8/-5`	four numbers: −8, −7, −6, −5
`-5/-8`	four numbers: −5, −6, −7, −8
`-1/2`	four numbers: −1, 0, 1, 2
`1 2 to 4`	four numbers: 1, 2, 3, 4
`4 3 to 1`	four numbers: 4, 3, 2, 1
`10 15 to 30`	five numbers: 10, 15, 20, 25, 30
`1 2:4`	same as 1 2 to 4
`4 3:1`	same as 4 3 to 1
`10 15:30`	same as 10 15 to 30
`1(1)3`	three numbers: 1, 2, 3
`1(2)9`	five numbers: 1, 3, 5, 7, 9
`1(2)10`	the same five numbers, 1, 3, 5, 7, 9
`9(-2)1`	five numbers: 9, 7, 5, 3, and 1
`-1(.5)2.5`	the numbers −1, −.5, 0, .5, 1, 1.5, 2, 2.5
`1[1]3`	same as 1(1)3
`1[2]9`	same as 1(2)9
`1[2]10`	same as 1(2)10
`9[-2]1`	same as 9(−2)1
`-1[.5]2.5`	same as −1(.5)2.5
`1 2 3/5 8(2)12`	eight numbers: 1, 2, 3, 4, 5, 8, 10, 12
`1,2,3/5,8(2)12`	the same eight numbers
`1 2 3/5 8 10 to 12`	the same eight numbers
`1,2,3/5,8,10 to 12`	the same eight numbers
`1 2 3/5 8 10:12`	the same eight numbers

`poisson`'s `constraints()` option has syntax `constraints(`*numlist*`)`. Thus you could type `constraints(2 4 to 8)`, `constraints(2(2)8)`, etc.

11.1.9 datelist

A *datelist* is a list of dates or times and is often used with graph options when the variable being graphed has a date format. For a description of how dates and times are stored and manipulated in Stata, see **[U] 25 Working with dates and times**. Calendar dates, also known as `%td` dates, are recorded in Stata as the number of days since 01jan1960, so 0 means 01jan1960, 1 means 02jan1960, and 16,541 means 15apr2005. Similarly, −1 means 31dec1959, −2 means 30dec1959, and −16,541 means 18sep1914. In such a case, a datelist is a list of dates, as in

```
15apr1973 17apr1973 20apr1973 23apr1973
```

or it is a first and last date with an increment between, as in

```
17apr1973(3)23apr1973
```

or it is a combination:

```
15apr1973 17apr1973(3)23apr1973
```

Dates specified with spaces, slashes, or commas must be bound in parentheses, as in

```
(15 apr 1973) (april 17, 1973)(3)(april 23, 1973)
```

Evenly spaced calendar dates are not especially useful, but with other time units, even spacing can be useful, such as

```
1999q1(1)2005q1
```

when `%tq` dates are being used. `1999q1(1)2005q1` means every quarter between 1999q1 and 2005q1. `1999q1(4)2005q1` would mean every first quarter.

To interpret a datelist, Stata first looks at the format of the related variable and then uses the corresponding date-to-numeric translation function. For instance, if the variable has a `%td` format, the `td()` function is used to translate the date; if the variable has a `%tq` format, the `tq()` function is used; and so on. See *Typing dates into expressions* in [D] **Datetime**.

11.1.10 Prefix commands

Stata has a handful of commands that are used to prefix other Stata commands. `by` *varlist*`:`, discussed in section [U] **11.1.2 by varlist:**, is in fact an example of a prefix command. In that section, we demonstrated `by` using

```
by region: summarize marriage_rate divorce_rate
```

and later,

```
by region, sort: summarize marriage_rate divorce_rate
```

and although we did not, we could also have demonstrated

```
by region, sort: summarize marriage_rate divorce_rate, detail
```

Each of the above runs the `summarize` command separately on the data for each region.

`by` itself follows standard Stata syntax:

`by` *varlist*[`,` *options*]`:` ...

In `by region, sort: summarize marriage_rate divorce_rate, detail`, `region` is `by`'s varlist and `sort` is `by`'s option, just as `marriage_rate` and `divorce_rate` are `summarize`'s varlist and `detail` is `summarize`'s option.

by is not the only prefix command, and the full list of such commands is

Prefix command	Description
by	run command on subsets of data
collect	run command and collect results to include in a table
frame	run command on the data in a specified frame
statsby	same as by, but collect statistics from each run
rolling	run command on moving subsets and collect statistics
bayesboot	perform Bayesian bootstrap estimation of specified statistics
bootstrap	run command on bootstrap samples
jackknife	run command on jackknife subsets of data
permute	run command on random permutations
simulate	run command on manufactured data
svy	run command and adjust results for survey sampling
mi estimate	run command on multiply imputed data and adjust results for multiple imputation (MI)
bayes	fit a Bayesian regression model
fmm	fit a finite mixture model
nestreg	run command with accumulated blocks of regressors and report nested model comparison tests
stepwise	run command with stepwise variable inclusion/exclusion
xi	run command after expanding factor variables and interactions; for most commands, using factor variables is preferred to using xi (see [U] **11.4.3 Factor variables**)
fp	run command with fractional polynomials of one regressor
mfp	run command with multiple fractional polynomial regressors
capture	run command and capture its return code
noisily	run command and show the output
quietly	run command and suppress the output
version	run command under specified version

The last group—capture, noisily, quietly, and version—deal with programming Stata and, for historical reasons, capture, noisily, and quietly allow you to omit the colon, so one programmer might code

```
quietly regress ...
```

and another

```
quietly: regress ...
```

All the other prefix commands require the colon. In addition to the corresponding reference manual entries, you may want to consult Baum (2016) for a richer discussion of prefix commands.

11.2 Abbreviation rules

Stata allows abbreviations. In this manual, we usually avoid abbreviating commands, variable names, and options to ensure readability:

```
. summarize myvar, detail
```

Experienced Stata users, on the other hand, tend to abbreviate the same command as

```
. sum myv, d
```

As a general rule, command, option, and variable names may be abbreviated to the shortest string of characters that uniquely identifies them.

This rule is violated if the command or option does something that cannot easily be undone; the command must then be spelled out in its entirety.

Also, a few common commands and options are allowed to have even shorter abbreviations than the general rule would allow.

The general rule is applied, without exception, to variable names.

11.2.1 Command abbreviation

The shortest allowed abbreviation for a command or option can be determined by looking at the command's syntax diagram. This minimal abbreviation is shown by underlining:

$\underline{\texttt{g}}\texttt{enerate}$
$\underline{\texttt{ap}}\texttt{pend}$
$\underline{\texttt{rot}}\texttt{ate}$
$\underline{\texttt{ru}}\texttt{n}$

If there is no underlining, no abbreviation is allowed. For example, `replace` may not be abbreviated, the underlying reason being that `replace` changes the data.

`rename` can be abbreviated `ren`, `rena`, or `renam`, or it can be spelled out in its entirety.

Sometimes short abbreviations are also allowed. Commands that begin with the letter *d* include `decode`, `describe`, `destring`, `dir`, `discard`, `display`, `do`, and `drop`, which suggests that the shortest allowable abbreviation for `describe` is `desc`. However, because `describe` is such a commonly used command, you may abbreviate it with the single letter `d`. You may also abbreviate the `list` command with the single letter `l`.

The other exception to the general abbreviation rule is that commands that alter or destroy data must be spelled out completely. Two commands that begin with the letter *d*, `discard` and `drop`, are destructive in the sense that, once you give one of these commands, there is no way to undo the result. Therefore, both must be spelled out.

The final exceptions to the general rule are commands implemented as ado-files. Such commands may not be abbreviated. Ado-file commands are external, and their names correspond to the names of disk files.

11.2.2 Option abbreviation

Option abbreviation follows the same logic as command abbreviation: you determine the minimum acceptable abbreviation by examining the command's syntax diagram. The syntax diagram for `summarize` reads, in part,

```
summarize ..., detail format
```

The `detail` option may be abbreviated `d`, `de`, `det`, ..., `detail`. Similarly, the `format` option may be abbreviated `f`, `fo`, ..., `format`.

The `clear` and `replace` options occur with many commands. The `clear` option indicates that even though completing this command will result in the loss of all data in memory, and even though the data in memory have changed since the data were last saved on disk, you want to continue. `clear` must be spelled out, as in `use newdata, clear`.

The `replace` option indicates that it is okay to save over an existing dataset. If you type `save mydata` and the file `mydata.dta` already exists, you will receive the message "file mydata.dta already exists", and Stata will refuse to overwrite it. To allow Stata to overwrite the dataset, you would type `save mydata, replace`. `replace` may not be abbreviated.

❑ Technical note

`replace` is a stronger modifier than `clear` and is one you should think about before using. With a mistaken `clear`, you can lose hours of work, but with a mistaken `replace`, you can lose days of work.

❑

11.2.3 Variable-name abbreviation

- Variable names may be abbreviated to the shortest string of characters that uniquely identifies them given the data currently loaded in memory.

  If your dataset contained four variables, `state`, `mrgrate`, `dvcrate`, and `dthrate`, you could refer to the variable `dvcrate` as `dvcrat`, `dvcra`, `dvcr`, `dvc`, or `dv`. You might type `list dv` to list the data on `dvcrate`. You could not refer to the variable `dvcrate` as `d`, however, because that abbreviation does not distinguish `dvcrate` from `dthrate`. If you were to type `list d`, Stata would respond with the message "ambiguous abbreviation". (If you wanted to refer to *all* variables that started with the letter *d*, you could type `list d*`; see [U] **11.4 varname and varlists**.)

- The character ~ may be used to mean that "zero or more characters go here". For instance, `r~8` might refer to the variable `rep78`, or `rep1978`, or `repair1978`, or just `r8`. (The ~ character is similar to the `*` character in [U] **11.4 varname and varlists**, except that it adds the restriction "and only one variable matches this specification".)

  Above, we said that you could abbreviate variables. You could type `dvcr` to refer to `dvcrate`, but, if there were more than one variable that started with the letters `dvcr`, you would receive an error. Typing `dvcr` is the same as typing `dvcr~`.

11.2.4 Abbreviations for programmers

Stata has several useful commands and functions to assist programmers with abbreviating and unabbreviating command names and variable names.

Command/function	Description
`unab`	expand and unabbreviate standard variable lists
`tsunab`	expand and unabbreviate variable lists that may contain time-series operators
`fvunab`	expand and unabbreviate variable lists that may contain time-series operators or factor variables
`unabcmd`	unabbreviate command name
`novarabbrev`	turn off variable abbreviation
`varabbrev`	turn on variable abbreviation
`set varabbrev`	set whether variable abbreviations are supported
`abbrev(`s`,`n`)`	string function that abbreviates s to n display columns
`abbrev(`s`,`n`)`	Mata variant of above that allows s and n to be matrices

11.3 Naming conventions

A name is a sequence of 1 to 32 letters (`A`–`Z`, `a`–`z`, and any Unicode letter), digits (`0`–`9`), and underscores (`_`).

Programmers: Local macro names can have no more than 31 characters in the name; see **[U] 18.3.1 Local macros**.

Stata reserves the following names:

```
alias    _n        _r_p
_all     _N        _r_se
_b       _pi       _r_ub
byte     _pred     _r_z
_coef    _r_b      _r_z_abs
_cons    _rc       _se
double   _r_ci     _skip
float    _r_cri    str#
if       _r_crlb   strL
in       _r_crub   using
int      _r_df     with
long     _r_lb
```

You may not use these reserved names for your variables.

The first character of a name must be a letter or an underscore (macro names are an exception; they may also begin with a digit). We recommend, however, that you not begin your variable names with an underscore. All of Stata's 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 three distinct names.

Most objects in Stata—not just variables—follow this naming convention.

11.4 varname and varlists

varlist is a list of variable names. The variable names in *varlist* refer either exclusively to new (not yet created) variables or exclusively to existing variables. *newvarlist* always refers exclusively to new (not yet created) variables. Similarly, *varname* refers to one variable, either existing or not yet created. *newvar* always refers to one new variable.

Sometimes a command will refer to a varname in another way, such as "*groupvar*". This is still a varname. The different name is used to give you an extra hint about the purpose of that variable. For example, *groupvar* is the name of a variable that defines groups within your data. Other common ways of referring to *varname* or *varlist* in Stata are

depvar, which means the dependent variable for an estimation command;

indepvars, which means *varlist* containing the independent variables for an estimation command;

xvar, which means a continuous real variable, often plotted on the x axis of a graph;

yvar, which means a variable that is a function of an *xvar*, often plotted on the y axis of a graph;

clustvar, which means a numeric variable that identifies the cluster or group to which an observation belongs;

panelvar, which means a numeric variable that identifies panels in panel data, also known as cross-sectional time-series data; and

timevar, which means a numeric variable with a `%td`, `%tc`, or `%tC` format.

11.4.1 Lists of existing variables

In lists of existing variable names, variable names may be repeated.

If you type `list state mrgrate dvcrate state`, the variable `state` will be listed twice, once in the leftmost column and again in the rightmost column of the list.

Existing variable names may be abbreviated as described in **[U] 11.2 Abbreviation rules**. You may also use "`*`" to indicate that "zero or more characters go here". For instance, if you suffix `*` to a partial variable name (for example, `sta*`), you are referring to all variable names that start with that letter combination. If you prefix `*` to a letter combination (for example, `*rate`), you are referring to all variables that end in that letter combination. If you put `*` in the middle (for example, `m*rate`), you are referring to all variables that begin and end with the specified letters. You may put more than one `*` in an abbreviation.

▷ Example 10

If the variables `poplt5`, `pop5to17`, and `pop18p` are in our dataset, we may type `pop*` as a shorthand way to refer to all three variables. For instance, `list state pop*` lists the variables `state`, `poplt5`, `pop5to17`, and `pop18p`.

If we had a dataset with variables `inc1990`, `inc1991`, ..., `inc1999` along with variables `incfarm1990`, ..., `incfarm1999`; `pop1990`, ..., `pop1999`; and `ms1990`, ..., `ms1999`, then `*1995` would be a shorthand way of referring to `inc1995`, `incfarm1995`, `pop1995`, and `ms1995`. We could type, for instance, `list *1995`.

In that same dataset, typing `list i*95` would be a shorthand way of listing `inc1995` and `incfarm1995`.

Typing `list i*f*95` would be a shorthand way of listing to `incfarm1995`.

◁

Typing ~ is an alternative to `*`, and really, it means the same thing. The difference is that ~ indicates that if more than one variable matches the specified pattern, Stata will complain rather than substituting all the variables that match the specification.

▷ Example 11

In the previous example, we could have typed `list i~f~95` to list `incfarm1995`. If, however, our dataset also included variable `infant1995`, then `list i*f*95` would list both variables and `list i~f~95` would complain that `i~f~95` is an ambiguous abbreviation.

◁

You may use `?` to specify that one character goes here. Remember, `*` means zero or more characters go here, so `?*` can be used to mean one or more characters goes here, `??*` can be used to mean two or more characters go here, and so on.

▷ Example 12

In a dataset containing variables `rep1`, `rep2`, ..., `rep78`, `rep?` would refer to `rep1`, `rep2`, ..., `rep9`, and `rep??` would refer to `rep10`, `rep11`, ..., `rep78`.

◁

You may place a dash (`-`) between two variable names to specify all the variables stored between the two listed variables, inclusive. You can determine storage order by using `describe`; it lists variables in the order in which they are stored.

▷ Example 13

If the dataset contains the variables `state`, `mrgrate`, `dvcrate`, and `dthrate`, in that order, typing `list state-dvcrate` is equivalent to typing `list state mrgrate dvcrate`. In both cases, three variables are listed.

◁

11.4.2 Lists of new variables

In lists of *new variables*, no variable names may be repeated or abbreviated.

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 variable names in ascending numerical order.

For example, typing `input v1-v4` is equivalent to typing `input v1 v2 v3 v4`. Typing `infile state v1-v3 ssn using rawdata` is equivalent to typing `infile state v1 v2 v3 ssn using rawdata`.

Many commands that require a specific number of new variables also allow the new variables to be specified using the *stub*`*` notation. For example, if you are using `predict` to generate four new variables, you could type `predict pred*` to create new variables `pred1`, `pred2`, `pred3`, and `pred4`.

You may specify the storage type before the variable name to force a storage type other than the default. The numeric storage types are `byte`, `int`, `long`, `float` (the default), and `double`. The string storage types are `str`*#*, where # is replaced with an integer between 1 and 2045, inclusive, representing the maximum length of the string, or `strL`. See [U] **12 Data**.

For instance, the list var1 str8 var2 var3 specifies that var1 and var3 be given the default storage type and that var2 be stored as a str8—a string whose maximum length is eight bytes.

The list var1 int var2 var3 specifies that var2 be stored as an int. You may use parentheses to bind a list of variable names. The list var1 int(var2 var3) specifies that both var2 and var3 be stored as ints. Similarly, the list var1 str20(var2 var3) specifies that both var2 and var3 be stored as str20s. The different storage types are listed in [U] **12.2.2 Numeric storage types** and [U] **12.4 Strings**.

▷ Example 14

Typing infile str2 state str10 region v1-v5 using mydata reads the state and region strings from the file mydata.raw and stores them as str2 and str10, respectively, along with the variables v1 through v5, which are stored as the default storage type float (unless we have specified a different default with the set type command).

Typing infile str10(state region) v1-v5 using mydata would achieve almost the same result, except that the state and region values recorded in the data would both be assigned to str10 variables. (We could then use the compress command to shorten the strings. See [D] **compress**; it is well worth reading.)

◁

❑ Technical note

You may append a colon and a *value label name* to numeric variables. (See [U] **12.6 Dataset, variable, and value labels** for a description of value labels.) For instance, var1 var2:myfmt specifies that the variable var2 be associated with the value label 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 value labels can then be assigned by the current command; see [D] **input** and [D] **infile (free format)**.

❑

▷ Example 15

Typing infile int(state:stfmt region:regfmt) v1-v5 using mydata, automatic reads the state and region data from the file mydata.raw and stores them as ints, along with the variables v1 through v5, which are stored as the default storage type.

In our previous example, both state and region were strings, so how can strings be stored in a numeric variable? See [U] **12.6 Dataset, variable, and value labels** for the complete answer. The colon notation specifies the name of the value label, and the automatic option tells Stata to assign unique numeric codes to all character strings. The numeric code for state, which Stata will make up on the fly, will be stored in the state variable. The mapping from numeric codes to words will be stored in the value label named stfmt. Similarly, regions will be assigned numeric codes, which are stored in region, and the mapping will be stored in regfmt.

If we were to list the data, the state and region variables would look like strings. state, for instance, would appear to contain things like AL, CA, and WA, but actually it would contain only numbers like 1, 2, 3, and 4.

◁

11.4.3 Factor variables

Factor variables are extensions of varlists of existing variables. When a command allows factor variables, in addition to typing variable names from your data, you can type factor variables, which might look like

`i.`*varname*

`i.`*varname*`#i.`*varname*

`i.`*varname*`#i.`*varname*`#i.`*varname*

`i.`*varname*`##i.`*varname*

`i.`*varname*`##i.`*varname*`##i.`*varname*

Factor variables create indicator variables from categorical variables and are allowed with most estimation and postestimation commands, along with a few other commands.

Consider a variable named `group` that takes on the values 1, 2, and 3. Stata command `list` allows factor variables, so we can see how factor variables are expanded by typing

```
. list group i.group in 1/5

     +-----------------------------------+
     |             1.       2.       3.  |
     | group    group    group    group  |
     |-----------------------------------|
  1. |     1        1        0        0  |
  2. |     1        1        0        0  |
  3. |     2        0        1        0  |
  4. |     2        0        1        0  |
  5. |     3        0        0        1  |
     +-----------------------------------+
```

There are no variables named `1.group`, `2.group`, and `3.group` in our data; there is only the variable named `group`. When we type `i.group`, however, Stata acts as if the variables `1.group`, `2.group`, and `3.group` exist. `1.group`, `2.group`, and `3.group` are called virtual variables. `1.group` is a virtual variable equal to 1 when `group` = 1, and 0 otherwise. `2.group` is a virtual variable equal to 1 when `group` = 2, and 0 otherwise. `3.group` is a virtual variable equal to 1 when `group` = 3, and 0 otherwise.

The categorical variable to which factor-variable operators are applied must contain nonnegative integers.

❏ Technical note

We said above that `3.group` equals 1 when `group` = 3 and equals 0 otherwise. We should have added that `3.group` equals missing when `group` contains missing. To be precise, `3.group` equals 1 when `group` = 3, equals system missing (`.`) when `group` $\geq$ `.`, and equals 0 otherwise.

❏

❏ Technical note

We said above that when we typed `i.group`, Stata acts as if the variables `1.group`, `2.group`, and `3.group` exist, and that might suggest that the act of typing `i.group` somehow created the virtual variables. That is not true; the virtual variables always exist.

In fact, `i.group` is an abbreviation for `1.group`, `2.group`, and `3.group`. In any command that allows factor variables, you can specify virtual variables. Thus the listing above could equally well have been produced by typing

```
. list group 1.group 2.group 3.group in 1/5
```

#.*varname* is defined as equal to 1 when *varname* = #, equal to system missing (.) when *varname* ≥ ., and equal to 0 otherwise. Thus 4.group is defined even when group takes on only the values 1, 2, and 3. 4.group would be equal to 0 in all observations. Referring to 4.group would not produce an error such as "virtual variable not found".

❑

When factor-variable operators are used in a regression command, one of the categories is chosen as a base category. If we type

```
. regress y i.group
```

this is equivalent to typing

```
. regress y 1b.group 2.group 3.group
```

1b.group is different from the other virtual variables. The b is a marker indicating base value. 1b.group is a virtual variable equal to 0. We can see this by typing

```
. list group i.group in 1/5

     +---------------------------------+
     |            1.       2.       3. |
     | group   group    group    group |
     |---------------------------------|
  1. |     1       1        0        0 |
  2. |     1       1        0        0 |
  3. |     2       0        1        0 |
  4. |     2       0        1        0 |
  5. |     3       0        0        1 |
     +---------------------------------+
```

When the i.group collection is included in a linear regression, virtual variable 1b.group drops from the estimation because it does not vary; thus the coefficients on 2.group and 3.group would measure the change from group = 1. Hence, the term base value.

11.4.3.1 Factor-variable operators

i.group is called a factor variable, although more correctly, we should say that group is a categorical variable to which factor-variable operators have been applied. There are five factor-variable operators:

Operator	Description
i.	unary operator to specify indicators
c.	unary operator to treat as continuous
o.	unary operator to omit a variable or indicator
#	binary operator to specify interactions
##	binary operator to specify full-factorial interactions

When you type i.group, it forms the indicators for the distinct values of group. We will usually say this more briefly as i.group forms indicators for the levels of group, and sometimes we will abbreviate the statement even more and say i.group forms indicators for group.

The c. operator means continuous. We will get to that below.

The o. operator specifies that a continuous variable or an indicator for a level of a categorical variable should be omitted. For example, o.age means that the continuous variable age should be omitted, and o2.group means that the indicator for group = 2 should be omitted.

# and ##, pronounced cross and factorial cross, are operators for use with pairs of variables.

i.group#i.sex means to form indicators for each combination of the levels of group and sex.

group#sex means the same thing, which is to say that use of # implies the i. prefix.

group#c.age (or i.group#c.age) means the interaction of the levels of group with the continuous variable age. This amounts to forming i.group and then multiplying each level by age. We already know that i.group expands to the virtual variables 1.group, 2.group, and 3.group, so group#c.age results in the collection of variables equal to 1.group*age, 2.group*age, and 3.group*age. 1.group*age will be age when group = 1, and 0 otherwise. 2.group*age will be age when group = 2, and 0 otherwise. 3.group*age will be age when group = 3, and 0 otherwise.

In a regression of y on age and group#c.age, group = 1 will again be chosen as the base value of group. Thus group#c.age expands to 1b.group*age, 2.group*age, and 3.group*age. 1b.group*age will be zero because 1b.group is zero, so it will be omitted. 2.group*age will measure the change in the age coefficient for group = 2 relative to the base group, and 3.group*age will measure the change for group = 3 relative to the base.

Here are some more examples of use of the operators:

Factor specification	Result
i.group	indicators for levels of group
i.group#i.sex	indicators for each combination of levels of group and sex, a two-way interaction
group#sex	same as i.group#i.sex
group#sex#arm	indicators for each combination of levels of group, sex, and arm, a three-way interaction
group##sex	same as i.group i.sex group#sex
group##sex##arm	same as i.group i.sex i.arm group#sex group#arm sex#arm group#sex#arm
sex#c.age	two variables—age for males and 0 elsewhere, and age for females and 0 elsewhere; if age is also in the model, one of the two virtual variables will be treated as a base
sex##c.age	same as i.sex age sex#c.age
c.age	same as age
c.age#c.age	age squared
c.age#c.age#c.age	age cubed

Several factor-variable terms are often specified in the same varlist, such as

```
. regress y  i.sex i.group sex#group age sex#c.age
```

or, equivalently,

```
. regress y  sex##group sex##c.age
```

11.4.3.2 Base levels

When we typed i.group in a regression command, group = 1 became the base level. When we do not specify otherwise, the smallest level becomes the base level.

You can specify the base level of a factor variable by using the ib. operator. The syntax is

Base operator[a]	Description
ib#.	use # as base, # = value of variable
ib(##).	use the #th ordered value as base[b]
ib(first).	use smallest value as base (default)
ib(last).	use largest value as base
ib(freq).	use most frequent value as base
ibn.	no base level

[a]The i may be omitted. For instance, you can type ib2.group or b2.group.
[b]For example, ib(#2). means to use the second value as the base.

Thus, if you want to use group = 3 as the base, you can type ib3.group. You can type

```
. regress y  i.sex ib3.group sex#ib3.group age sex#c.age
```

or you can type

```
. regress y  i.sex ib3.group sex#group age sex#c.age
```

That is, you only have to set the base once. If you specify the base level more than once, it must be the same base level. You will get an error if you attempt to change base levels in midsentence.

If you type ib3.group, the virtual variables become 1.group, 2.group, and 3b.group.

Were you to type ib(freq).group, the virtual variables might be 1b.group, 2.group, and 3.group; 1.group, 2b.group, and 3.group; or 1.group, 2.group, and 3b.group, depending on the most frequent group in the data.

11.4.3.3 Setting base levels permanently

You can permanently set the base level by using the fvset command; see [R] **fvset**. For example,

```
. fvset base 3  group
```

sets the base for group to be 3. The setting is recorded in the data, and if the dataset is resaved, the base level will be remembered in future sessions.

If you want to set the base group back to the default, type

```
. fvset base default  group
```

If you want to set the base levels for a group of variables to be the largest value, you can type

```
. fvset base last  group sex arm
```

See [R] **fvset** for details.

Base levels can be temporarily overridden by using the ib. operator regardless of whether they are set explicitly.

11.4.3.4 Selecting levels

Typing `i.group` specifies virtual variables `1b.group`, `2.group`, and `3.group`. Regardless of whether you type `i.group`, you can access those virtual variables. You can, for instance, use them in expressions and `if` statements:

```
. list if 3.group
  (output omitted)
. generate over_age = cond(3.group, age-21, 0)
```

Although throughout this section we have been typing `#.group` such as `3.group` as if it is somehow different from `i.group`, the complete, formal syntax is `i3.group`. You are allowed to omit the `i`. The point is that `i3.group` is just a special case of `i.group`; `i3.group` specifies an indicator for the third level of `group`, and `i.group` specifies the indicators for all the levels of `group`. Anyway, the above commands could be typed as

```
. list if i3.group
  (output omitted)
. generate over_age = cond(i3.group, age-21, 0)
```

Similarly, the virtual variables `1b.group`, `2.group`, and `3.group` more formally would be referred to as `i1b.group`, `i2.group`, and `i3.group`. You are allowed to omit the leading `i` whenever what appears after is a number or a `b` followed by a base specification.

You can select a range of levels—a range of virtual variables—by using the `i(`*numlist*`).`*varname*. This can be useful when specifying the model to be fit using estimation commands. You may not omit the `i` when specifying a numlist.

Examples	Description
`i2.cat`	single indicator for `cat` = 2
`2.cat`	same as `i2.cat`
`i(2 3 4).cat`	three indicators, `cat` = 2, `cat` = 3, and `cat` = 4; same as `i2.cat i3.cat i4.cat`
`i(2/4).cat`	same as `i(2 3 4).cat`
`2.cat#1.sex`	a single indicator that is 1 when `cat` = 2 and `sex` = 1 and is 0 otherwise
`i2.cat#i1.sex`	same as `2.cat#1.sex`

Rather than selecting the levels that should be included, you can specify the levels that should be omitted by using the `o.` operator. When you use `io(`*numlist*`).`*varname* in a command, indicators for the levels of *varname* other than those specified in *numlist* are included. When omitted levels are specified with the `o.` operator, the `i.` operator is implied, and the remaining indicators for the levels of *varname* will be included.

Examples	Description
`io2.cat`	indicators for levels of `cat`, omitting the indicator for `cat` = 2
`o2.cat`	same as `io2.cat`
`io(2 3 4).cat`	indicators for levels of `cat`, omitting three indicators, `cat` = 2, `cat` = 3, and `cat` = 4
`o(2 3 4).cat`	same as `io(2 3 4).cat`
`o(2/4).cat`	same as `io(2 3 4).cat`
`o2.cat#o1.sex`	indicators for each combination of the levels of `cat` and `sex`, omitting the indicator for `cat` = 2 and `sex` = 1

11.4.3.5 Applying operators to a group of variables

Factor-variable operators may be applied to groups of variables by using parentheses. You may type, for instance,

```
i.(group sex arm)
```

to mean `i.group i.sex i.arm`.

In the examples that follow, variables `group`, `sex`, `arm`, and `cat` are categorical, and variables `age`, `wt`, and `bp` are continuous:

Examples	Expansion
`i.(group sex arm)`	`i.group i.sex i.arm`
`group#(sex arm cat)`	`group#sex group#arm group#cat`
`group##(sex arm cat)`	`i.group i.sex i.arm i.cat group#sex group#arm group#cat`
`group#(c.age c.wt c.bp)`	`group#c.age group#c.wt group#c.bp`
`group#c.(age wt bp)`	same as `group#(c.age c.wt c.bp)`

Parentheses can shorten what you type and make it more readable. For instance,

```
. regress y  i.sex i.group sex#group age sex#c.age c.age#c.age sex#c.age#c.age
```

is easier to understand when written as

```
. regress y  sex##(group c.age c.age#c.age)
```

11.4.3.6 Using factor variables with time-series operators

Factor-variable operators may be combined with the `L.` and `F.` time-series operators, so you may specify lags and leads of factor variables in time-series applications. You could type `iL.group` or `Li.group`; the order of the operators does not matter. You could type `L.group#L.arm` or `L.group#c.age`.

Examples include

```
. regress y b1.sex##(i(2/4).group cL.age cL.age#cL.age)
. regress y 2.arm#(sex#i(2/4)b3.group cL.age)
. regress y 2.arm##cat##(sex##i(2/4)b3.group cL.age#c.age) c.bp
>           c.bp#c.bp c.bp#c.bp#c.bp sex##c.bp#c.age
```

11.4.3.7 Video examples

Introduction to factor variables in Stata, part 1: The basics

Introduction to factor variables in Stata, part 2: Interactions

Introduction to factor variables in Stata, part 3: More interactions

11.4.4 Time-series varlists

Time-series varlists are a variation on varlists of existing variables. When a command allows a time-series varlist, you may include time-series operators. For instance, `L.gnp` refers to the lagged value of variable gnp. The time-series operators are

Operator	Meaning
`L.`	lag x_{t-1}
`L2.`	2-period lag x_{t-2}
...	
`F.`	lead x_{t+1}
`F2.`	2-period lead x_{t+2}
...	
`D.`	difference $x_t - x_{t-1}$
`D2.`	difference of difference $x_t - x_{t-1} - (x_{t-1} - x_{t-2}) = x_t - 2x_{t-1} + x_{t-2}$
...	
`S.`	"seasonal" difference $x_t - x_{t-1}$
`S2.`	lag-2 (seasonal) difference $x_t - x_{t-2}$
...	

Time-series operators may be repeated and combined. `L3.gnp` refers to the third lag of variable gnp. So do `LLL.gnp`, `LL2.gnp`, and `L2L.gnp`. `LF.gnp` is the same as gnp. `DS12.gnp` refers to the one-period difference of the 12-period difference. `LDS12.gnp` refers to the same concept, lagged once.

`D1.` = `S1.`, but `D2.` ≠ `S2.`, `D3.` ≠ `S3.`, and so on. `D2.` refers to the difference of the difference. `S2.` refers to the two-period difference. If you wanted the difference of the difference of the 12-period difference of gnp, you would write `D2S12.gnp`.

Operators may be typed in uppercase or lowercase. Most users would type `d2s12.gnp` instead of `D2S12.gnp`.

You may type operators however you wish; Stata internally converts operators to their canonical form. If you typed `ld2ls12d.gnp`, Stata would present the operated variable as `L2D3S12.gnp`.

In addition to using *operator#*, Stata understands *operator*`(`*numlist*`)` to mean a set of operated variables. For instance, typing `L(1/3).gnp` in a varlist is the same as typing `L.gnp L2.gnp L3.gnp`. The operators can also be applied to a list of variables by enclosing the variables in parentheses; for example,

```
. use https://www.stata-press.com/data/r19/gxmpl1
. list year L(1/3).(gnp cpi)

     +-----------------------------------------------------------------+
     |              L.       L2.       L3.        L.       L2.      L3.|
     | year        gnp       gnp       gnp       cpi       cpi      cpi|
     |-----------------------------------------------------------------|
  1. | 1989          .         .         .         .         .        .|
  2. | 1990     5837.9         .         .       124         .        .|
  3. | 1991     6026.3    5837.9         .     130.7       124        .|
  4. | 1992     6367.4    6026.3    5837.9     136.2     130.7      124|
  5. | 1993     6689.3    6367.4    6026.3     140.3     136.2    130.7|
     |-----------------------------------------------------------------|
  6. | 1994     7098.4    6689.3    6367.4     144.5     140.3    136.2|
  7. | 1995     7433.4    7098.4    6689.3     148.2     144.5    140.3|
  8. | 1996     7851.9    7433.4    7098.4     152.4     148.2    144.5|
     +-----------------------------------------------------------------+
```

The parentheses notation may be used with any operator. Typing `D(1/3).gnp` would return the first through third differences.

The parentheses notation may be used in operator lists with multiple operators, such as `L(0/3)D2S12.gnp`.

Operator lists may include up to one set of parentheses, which may enclose a *numlist*; see **[U] 11.1.8 numlist**.

The time-series operators `L.` and `F.` may be combined with factor variables. If we want to lag the indicator variables for the levels of the factor variable `region`, we would type `iL.region`. We could also say that we are specifying the level indicator variables for the lag of the region variables. They are equivalent statements.

The numlists and parentheses notation from both factor varlists and time-series operators may be combined. For example, `iL(1/3).region` specifies the first three lags of the level indicators for `region`. If `region` has four levels, this is equivalent to typing `i1L1.region i2L1.region i3L1.region i4L1.region i1L2.region i2L2.region i3L2.region i4L2.region i1L3.region i2L3.region i3L3.region i4L3.region`. Pushing the notation further, `i(1/2)L(1/3).(region education)` specifies the first three lags of the level 1 and level 2 indicator variables for both `region` and `education`.

❑ Technical note

The `D.` and `S.` time-series operators may not be combined with factor variables because such combinations could have two meanings. `iD.a` could be the level indicators for the difference of the variable `a` from its prior period, or it could be the level indicators differenced between the two periods. These are generally not the same values, nor even the same number of indicators. Moreover, they are rarely interesting.

❑

Before you can use time-series operators in varlists, you must set the time variable by using the `tsset` command:

```
. list l.gnp
time variable not set
r(111);

. tsset time
  (output omitted)

. list l.gnp
  (output omitted)
```

See **[TS] tsset**. The time variable must take on integer values. Also, the data must be sorted on the time variable. `tsset` handles this, but later you might encounter

```
. list l.mpg
not sorted
r(5);
```

Then type `sort time` or type `tsset` to reestablish the order.

The time-series operators respect the time variable. `L2.gnp` refers to $\texttt{gnp}_{t-2}$, regardless of missing observations in the dataset. In the following dataset, the observation for 1992 is missing:

```
. use https://www.stata-press.com/data/r19/gxmpl2

. list year gnp l2.gnp, separator(0)

     ┌─────────────────────────┐
     │                     L2. │
     │ year      gnp      gnp  │
     ├─────────────────────────┤
  1. │ 1989   5837.9        .  │
  2. │ 1990   6026.3        .  │
  3. │ 1991   6367.4   5837.9  │
  4. │ 1993   7098.4   6367.4  │      ← note, filled in correctly
  5. │ 1994   7433.4        .  │
  6. │ 1995   7851.9   7098.4  │
     └─────────────────────────┘
```

Operated variables may be used in expressions:

```
. generate gnplag2 = l2.gnp
(3 missing values generated)
```

Stata also understands cross-sectional time-series data. If you have cross sections of time series, you indicate this when you `tsset` the data:

```
. tsset country year
```

See [TS] **tsset**. In fact, you can type that, or you can type

```
. xtset country year
```

`xtset` is how you set panel data just as `tsset` is how you set time-series data and here the two commands do the same thing. Some panel datasets are not cross-sectional time series, however, in that the second variable is not time, so `xtset` also allows

```
. xtset country
```

See [XT] **xtset**.

11.4.4.1 Video example

Time series, part 3: Time-series operators

11.5 by varlist: construct

by *varlist*: *command*

The by prefix causes *command* to be repeated for each distinct value or combination of values of the variables in *varlist*. *varlist* may contain numeric, string, or a mixture of numeric and string variables. (*varlist* may not contain time-series operators.)

by is an optional prefix to perform a Stata command separately for each group of observations where the values of the variables in the *varlist* are the same.

During each iteration, the values of the system variables _n and _N are set in relation to the first observation in the by-group; see [U] **13.7 Explicit subscripting**. The in *range* qualifier cannot be used with by *varlist*: because ranges specify absolute rather than relative observation numbers.

❑ Technical note

The inability to combine in and by is not really a constraint because if provides all the functionality of in and a bit more. If you wanted to perform *command* for the first three observations in each of the by-groups, you could type

```
. by varlist: command if _n<=3
```

❑

The results of *command* would be the same as if you had formed separate datasets for each group of observations, saved them, used each separately, and issued *command*.

▷ Example 16

We provide some examples using by in [U] **11.1.2 by varlist:** above. We demonstrate the effect of by on _n, _N, and explicit subscripting in [U] **13.7 Explicit subscripting**.

by requires that the data first be sorted. For instance, if we had data on the average January and July temperatures in degrees Fahrenheit for 420 cities located in the Northeast and West and wanted to obtain the averages, by region, across those cities, we might type

```
. use https://www.stata-press.com/data/r19/citytemp3, clear
(City temperature data)
. by region: summarize tempjan tempjuly
not sorted
r(5);
```

Stata refused to honor our request because the data are not sorted by region. We must either sort the data by region first (see [D] **sort**) or specify by's sort option (which has the same effect):

```
. by region, sort: summarize tempjan tempjuly

-------------------------------------------------------------------------------
-> region = NE

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
     tempjan |        164    27.88537    3.543096       16.6       31.8
    tempjuly |        164       73.35    2.361203       66.5       76.8

-------------------------------------------------------------------------------
  (output omitted)
-------------------------------------------------------------------------------
-> region = West

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
     tempjan |        256    46.22539    11.25412         13       72.6
    tempjuly |        256    72.10859    6.483131       58.1       93.6
```

◁

▷ Example 17

Using the same data as in the example above, we estimate regressions, by `region`, of average January temperature on average July temperature. Both temperatures are specified in degrees Fahrenheit.

```
. by region: regress tempjan tempjuly

-----------------------------------------------------------------------------
-> region = NE
      Source |       SS           df       MS      Number of obs   =       164
-------------+----------------------------------   F(1, 162)       =    479.82
       Model |  1529.74026         1  1529.74026   Prob > F        =    0.0000
    Residual |  516.484453       162  3.18817564   R-squared       =    0.7476
-------------+----------------------------------   Adj R-squared   =    0.7460
       Total |  2046.22471       163  12.5535258   Root MSE        =    1.7855

------------------------------------------------------------------------------
     tempjan | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
    tempjuly |   1.297424   .0592303    21.90   0.000     1.180461    1.414387
       _cons |  -67.28066   4.346781   -15.48   0.000    -75.86431     -58.697
------------------------------------------------------------------------------

  (output omitted)
-----------------------------------------------------------------------------
-> region = West
      Source |       SS           df       MS      Number of obs   =       256
-------------+----------------------------------   F(1, 254)       =      2.84
       Model |  357.161728         1  357.161728   Prob > F        =    0.0932
    Residual |  31939.9031       254   125.74765   R-squared       =    0.0111
-------------+----------------------------------   Adj R-squared   =    0.0072
       Total |  32297.0648       255  126.655156   Root MSE        =    11.214

------------------------------------------------------------------------------
     tempjan | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
    tempjuly |   .1825482   .1083166     1.69   0.093    -.0307648    .3958613
       _cons |    33.0621    7.84194     4.22   0.000     17.61859     48.5056
------------------------------------------------------------------------------
```

The regressions show that a 1-degree increase in the average July temperature in the Northeast corresponds to a 1.3-degree increase in the average January temperature. In the West, however, it corresponds to a 0.18-degree increase, which is only marginally significant.

◁

❑ Technical note

by has a second syntax that is especially useful when you want to play it safe:

by *varlist*$_1$ (*varlist*$_2$) : *command*

This says that Stata is to verify that the data are sorted by *varlist*$_1$ *varlist*$_2$ and then, assuming that is true, perform *command* by *varlist*$_1$. For instance,

```
. by subject (time): generate finalval = val[_N]
```

By typing this, we want to create new variable `finalval`, which contains, in each observation, the final observed value of `val` for each subject in the data. The final value will be the last value if, within subject, the data are sorted by time. The above command verifies that the data are sorted by `subject` and `time` and then, if they are, performs

```
. by subject: generate finalval = val[_N]
```

If the data are not sorted properly, an error message will instead be issued. Of course, we could have just typed

```
. by subject: generate finalval = val[_N]
```

after verifying for ourselves that the data were sorted properly, as long as we were careful to look.

by's second syntax can be used with by's `sort` option, so we can also type

```
. by subject (time), sort: generate finalval = val[_N]
```

which is equivalent to

```
. sort subject time
. by subject: generate finalval = val[_N]
```

❑

See Mitchell (2020, chap. 8) for numerous examples of processing groups using the `by:` construct. Also see Cox (2002).

11.6 Filenaming conventions

Some commands require that you specify a *filename*. Filenames are specified in the way natural for your operating system:

Windows	Unix	Mac
`mydata`	`mydata`	`mydata`
`mydata.dta`	`mydata.dta`	`mydata.dta`
`c:mydata.dta`	`~friend/mydata.dta`	`~friend/mydata.dta`
`"my data"`	`"my data"`	`"my data"`
`"my data.dta"`	`"my data.dta"`	`"my data.dta"`
`myproj\mydata`	`myproj/mydata`	`myproj/mydata`
`"my project\my data"`	`"my project/my data"`	`"my project/my data"`
`C:\analysis\data\mydata`	`~/analysis/data/mydata`	`~/analysis/data/mydata`
`"C:\my project\my data"`	`"~/my project/my data"`	`"~/my project/my data"`
`..\data\mydata`	`../data/mydata`	`../data/mydata`
`"..\my project\my data"`	`"../my project/my data"`	`"../my project/my data"`

We strongly discourage using Unicode characters beyond plain ASCII in filenames because different operating systems use different UTF encodings for Unicode characters. For example, because Linux encodes filenames in UTF-8 and Windows encodes them in UTF-16, the file may become unusable after it has been transferred from one system to another if it contains Unicode characters beyond plain ASCII.

In most cases, where *filename* is a file that you are loading, *filename* may also be a URL. For instance, we might specify use `https://www.stata-press.com/data/r19/nlswork`.

All operating systems allow blanks in filenames, and so does Stata. However, if the filename includes a blank, you must enclose the filename in double quotes. Typing

```
. save "my data"
```

would create the file `my data.dta`. Typing

```
. save my data
```

would be an error.

Usually (the exceptions being `copy`, `dir`, `ls`, `erase`, `rm`, and `type`), Stata automatically provides a file extension if you do not supply one. For instance, if you type `use mydata`, Stata assumes that you mean `use mydata.dta` because `.dta` is the file extension Stata normally uses for data files.

Stata provides the following default file extensions that are used by various commands:

`.ado`	automatically loaded do-files
`.dct`	text data dictionary
`.do`	do-file
`.dot`	decision tree plot file
`.dta`	Stata dataset file format
`.dtas`	Stata frameset file format
`.dtasig`	`datasignature` file
`.gph`	`graph`
`.grec`	Graph Editor recording (text format)
`.irf`	impulse–response function datasets
`.log`	log file in text format
`.mata`	Mata source code
`.mlib`	Mata library
`.mmat`	Mata matrix
`.mo`	Mata object file
`.raw`	text-format data
`.smcl`	log file in SMCL format
`.stbcal`	business calendars
`.ster`	saved estimates
`.stgrf`	ancillary file to `.ster` when using the `cate` command
`.sthlp`	help file
`.stjson`	Stata collection results, labels, and styles
`.stpr`	project file
`.stptrace`	parameter-trace file; see [MI] **mi ptrace**
`.stsem`	SEM Builder file
`.stswm`	spatial weighting matrix
`.stswp`	Do-file Editor backup (swap) file
`.stxer`	ancillary file to `.ster` when using lasso commands
`.sum`	checksum files to verify network transfers

You do not have to name your data files with the `.dta` extension—if you type an explicit file extension, it will override the default. For instance, if your dataset was stored as `myfile.dat`, you could type `use myfile.dat`. If your dataset was stored as simply `myfile` with no file extension, you could type the period at the end of the filename to indicate that you are explicitly specifying the null extension. You would type `use myfile.` to use this dataset.

❑ Technical note

Stata also uses other file extensions. These files are of interest only to advanced programmers or are for Stata's internal use. They are

`.class`	class file for object-oriented programming; see [P] **class**
`.dlg`	dialog resource file
`.idlg`	dialog resource include file
`.ihlp`	help include file
`.key`	`search`'s keyword database file
`.maint`	maintenance file (for Stata's internal use only)
`.mnu`	menu file (for Stata's internal use only)
`.pkg`	user-site package file
`.plugin`	compiled addition (DLL)
`.scheme`	control file for a graph scheme
`.style`	graph style file
`.toc`	user-site description file

❑

11.6.1 A special note for Mac users

Have you seen the notation `myfolder/myfile` before? This notation is called a path and describes the location of a file or folder (also called a directory).

You do not have to use this notation if you do not like it. You could instead restrict yourself to using files only in the current folder. If that turns out to be too restricting, Stata for Mac provides enough menus and buttons that you can probably get by. You may, however, find the notation convenient. If you do, here is the rest of the definition.

The character `/` is called a path delimiter and delimits folder names and filenames in a path. If the path starts with no path delimiter, the path is relative to the current folder.

For example, the path `myfolder/myfile` refers to the file `myfile` in the folder `myfolder`, which is contained in the current folder.

The characters `..` refer to the folder containing the current folder. Thus `../myfile` refers to `myfile` in the folder containing the current folder, and `../nextdoor/myfile` refers to `myfile` in the folder `nextdoor` in the folder containing the current folder.

If a path starts with a path delimiter, the path is called an absolute path and describes a fixed location of a file or folder name, regardless of what the current folder is. The leading `/` in an absolute path refers to the root directory, which is the main hard drive from which the operating system is booted. For example, the path `/myfolder/myfile` refers to the file `myfile` in the folder `myfolder`, which is contained in the main hard drive.

11.6.2 A shortcut to your home directory

Stata understands ~ to mean your home directory. Thus, you can refer to a dataset named `mydata.dta` in a subdirectory named `mydir` within your home directory by referring to the path

```
~\mydir\mydata.dta
```

in Stata for Windows or by referring to the path

```
~/mydir/mydata.dta
```

in Stata for Mac or Stata for Unix.

11.7 References

Baum, C. F. 2016. *An Introduction to Stata Programming*. 2nd ed. College Station, TX: Stata Press.

Buis, M. L. 2020. Stata tip 135: Leaps and bounds. *Stata Journal* 20: 244–249.

Cox, N. J. 2002. Speaking Stata: How to move step by: step. *Stata Journal* 2: 86–102.

———. 2009. Stata tip 79: Optional arguments to options. *Stata Journal* 9: 504.

———. 2023. Stata tip 151: Puzzling out some logical operators. *Stata Journal* 23: 293–297.

Cox, N. J., and C. B. Schechter. 2019. Speaking Stata: How best to generate indicator or dummy variables. *Stata Journal* 19: 246–259.

———. 2023. Stata tip 152: if and if: When to use the if qualifier and when to use the if command. *Stata Journal* 23: 589–594.

Daniels, L., and N. Minot. 2020. *An Introduction to Statistics and Data Analysis Using Stata*. Thousand Oaks, CA: Sage.

Kolev, G. I. 2006. Stata tip 31: Scalar or variable? The problem of ambiguous names. *Stata Journal* 6: 279–280.

Mitchell, M. N. 2020. *Data Management Using Stata: A Practical Handbook*. 2nd ed. College Station, TX: Stata Press.

Ryan, P. 2005. Stata tip 22: Variable name abbreviation. *Stata Journal* 5: 465–466.

12 Data

Contents

12.1 Data and datasets

Data form a rectangular table of numeric and string values in which each row is an observation on all the variables and each column contains the observations on one variable. Variables are designated by variable names. Observations are numbered sequentially from `1` to `_N`. The following example of data contains the first five odd and first five even positive integers, along with a string variable:

```
        odd  even   name
   1.     1     2   Bill
   2.     3     4   Mary
   3.     5     6    Pat
   4.     7     8  Roger
   5.     9    10   Sean
```

The observations are numbered `1` to `5`, and the variables are named `odd`, `even`, and `name`. Observations are referred to by number, and variables by name.

A dataset is data plus labelings, formats, notes, and characteristics.

All aspects of data and datasets are defined here. Long (2009) offers a long-time Stata user's hard-won advice on how to manage data in Stata to promote accurate, replicable research. Mitchell (2020) provides many examples on data management in Stata.

12.2 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`). The following are examples of valid numbers:

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

❑ Technical note

Stata also allows numbers to be represented in a hexadecimal/binary format, defined as

[`+`|`-`]`0.0`[⟨*zeros*⟩]{`X`|`x`}`-3ff`

or

[`+`|`-`]`1.`⟨*hexdigit*⟩[⟨*hexdigits*⟩]{`X`|`x`}{`+`|`-`}⟨*hexdigit*⟩[⟨*hexdigits*⟩]

The lead digit is always 0 or 1; it is 0 only when the number being expressed is zero. A maximum of 13 digits to the right of the hexadecimal point are allowed. The power ranges from `-3ff` to `+3ff`. The number is expressed in hexadecimal (base 16) digits; the number a`X+`b means $a \times 2^b$. For instance, `1.0X+3` is 2^3 or 8. `1.8X+3` is 12 because 1.8_{16} is $1 + 8/16 = 1.5$ in decimal and the number is thus $1.5 \times 2^3 = 1.5 \times 8 = 12$.

Stata can also display numbers using this format; see **[U] 12.5.1 Numeric formats**. For example,

```
. display 1.81x+2
6.015625

. display %21x 6.015625
+1.8100000000000X+002
```

This hexadecimal format is of special interest to numerical analysts.

❑

12.2.1 Missing values

A number may also take on the special value missing, denoted by a period (`.`). You specify a missing value anywhere that you may specify a number. Missing values differ from ordinary numbers in one respect: any arithmetic operation on a missing value yields a missing value.

In fact, there are 27 missing values in Stata: '`.`', the one just discussed, as well as `.a`, `.b`, …, and `.z`, which are known as extended missing values. The missing value '`.`' is known as the default or system missing value. Some people use extended missing values to indicate why a certain value is unknown—the question was not asked, the person refused to answer, etc. Other people have no use for extended missing values and just use '`.`'.

Stata's default or system missing value will be returned when you perform an arithmetic operation on missing values or when the arithmetic operation is not defined, such as division by zero, or the logarithm of a nonpositive number.

```
. display 2/0
.

. list

     +----+
     |  a |
     |----|
  1. | .b |
  2. |  . |
  3. | .a |
  4. |  3 |
  5. |  6 |
     +----+

. generate x = a + 1
(3 missing values generated)

. list

     +---------+
     |  a    x |
     |---------|
  1. | .b    . |
  2. |  .    . |
  3. | .a    . |
  4. |  3    4 |
  5. |  6    7 |
     +---------+
```

Numeric missing values are represented by "large positive values". The ordering is

$$\text{all numbers} < \texttt{.} < \texttt{.a} < \texttt{.b} < \cdots < \texttt{.z}$$

Thus the expression

$$\text{age} > 60$$

is true if variable age is greater than 60 or is missing. Similarly,

$$\texttt{gender} \neq 0$$

is true if `gender` is not zero or is missing.

To exclude missing values, you must ask whether the value is less than ‘`.`’; to detect missing values, you must ask whether the value is greater than or equal to ‘`.`’. For instance,

```
. list if age>60 & age<.

. generate agegt60 = 0 if age<=60

. replace agegt60 = 1 if age>60 & age<.

. generate agegt60 = (age>60) if age<.
```

❑ Technical note

Before Stata 8, Stata had only one representation for missing values, the period (`.`).

To ensure that old programs and do-files continue to work properly, when `version` is set less than 8, all missing values are treated as being the same. Thus `. == .a == .b == .z`, and so ‘*exp*`==.`’ and ‘*exp*`!=.`’ work just as they previously did.

❑

▷ Example 1

We have data on the income of husbands and wives recorded in the variables `hincome` and `wincome`, respectively. Typing the `list` command, we see that your data contain

```
. use https://www.stata-press.com/data/r19/gxmpl3

. list

     ┌───────────────────┐
     │ hincome   wincome │
     ├───────────────────┤
  1. │   32000         0 │
  2. │   35000     34000 │
  3. │   47000        .b │
  4. │      .z     50000 │
  5. │      .a         . │
     └───────────────────┘
```

The values of `wincome` in the third and fifth observations are missing, as distinct from the value of `wincome` in the first observation, which is known to be zero.

If we use the `generate` command to create a new variable, `income`, that is equal to the sum of `hincome` and `wincome`, three missing values would be produced.

```
. generate income = hincome + wincome
(3 missing values generated)

. list

     ┌────────────────────────────┐
     │ hincome   wincome   income │
     ├────────────────────────────┤
  1. │   32000         0    32000 │
  2. │   35000     34000    69000 │
  3. │   47000        .b        . │
  4. │      .z     50000        . │
  5. │      .a         .        . │
     └────────────────────────────┘
```

`generate` produced a warning message that 3 missing values were created, and when we list the data, we see that 47,000 plus missing yields missing.

◁

❑ Technical note

Stata stores numeric missing values as the largest 27 numbers allowed by the particular storage type; see [U] **12.2.2 Numeric storage types**. There are two important implications. First, if you `sort` on a variable that has missing values, the missing values will be placed last, and the sort order of any missing values will follow the rule regarding the properties of missing values stated above.

```
. sort wincome

. list wincome

     +---------+
     | wincome |
     |---------|
  1. |       0 |
  2. |   34000 |
  3. |   50000 |
  4. |       . |
  5. |      .b |
     +---------+
```

The second implication concerns relational operators and missing values. Do not forget that a missing value will be larger than any numeric value.

```
. list if wincome > 40000

     +------------------------------+
     | hincome   wincome     income |
     |------------------------------|
  3. |      .z     50000          . |
  4. |      .a         .          . |
  5. |   47000        .b          . |
     +------------------------------+
```

Observations 4 and 5 are listed because ‘`.`’ and ‘`.b`’ are both missing and thus are greater than 40,000. Relational operators are discussed in detail in [U] **13.2.3 Relational operators**.

❑

▷ Example 2

In producing statistical output, Stata ignores observations with missing values. Continuing with the example above, if we request summary statistics on `hincome` and `wincome` by using the `summarize` command, we obtain

```
. summarize hincome wincome

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
     hincome |          3       38000    7937.254      32000      47000
     wincome |          3       28000    25534.29          0      50000
```

Some commands discard the entire observation (known as casewise deletion) if one of the variables in the observation is missing. If we use the `correlate` command to obtain the correlation between `hincome` and `wincome`, for instance, we obtain

```
. correlate hincome wincome
(obs=2)

             |  hincome  wincome
-------------+------------------
     hincome |   1.0000
     wincome |   1.0000   1.0000
```

The correlation coefficient is calculated over two observations.

◁

12.2.2 Numeric storage types

Numbers can be stored in one of five variable types: `byte`, `int`, `long`, `float` (the default), or `double`. `bytes` are, naturally, stored in 1 byte. `ints` are stored in 2 bytes, `longs` and `floats` in 4 bytes, and `doubles` in 8 bytes. The table below shows the minimum and maximum values for each storage type.

Storage type	Minimum	Maximum	Closest to 0 without being 0	Bytes
`byte`	-127	100	± 1	1
`int`	$-32{,}767$	32,740	± 1	2
`long`	$-2{,}147{,}483{,}647$	2,147,483,620	± 1	4
`float`	$-1.70141173319 \times 10^{38}$	$1.70141173319 \times 10^{38}$	$\pm 10^{-38}$	4
`double`	$-8.9884656743 \times 10^{307}$	$+8.9884656743 \times 10^{307}$	$\pm 10^{-323}$	8

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

12.3 Dates and times

Stata has nine date, time, and date-and-time numeric encodings known collectively as `%t` variables or values. They are

`%tC`	calendar date and time, adjusted for leap seconds
`%tc`	calendar date and time, ignoring leap seconds
`%td`	calendar date
`%tw`	week
`%tm`	calendar month
`%tq`	financial quarter
`%th`	financial half-year
`%ty`	calendar year
`%tb`	business calendars

All except `%ty` and `%tb` are based on 0 = beginning of January 1960. `%tc` and `%tC` record the number of milliseconds since then. `%td` records the number of days. The others record the numbers of weeks, months, quarters, or half-years. `%ty` simply records the year, and `%tb` records a user-defined business calendar format.

For a full discussion of working with dates and times, see **[U] 25 Working with dates and times**.

12.4 Strings

This section describes the treatment of strings by Stata. The section is divided into the following subsections:

12.4.1 Overview

A string is a sequence of characters.

```
Samuel Smith
California
UK
```

Usually—but not always—strings are enclosed in double quotes.

```
"Samuel Smith"
"California"
"UK"
```

Strings typed in quotes are called string literals.

Strings can be stored in Stata datasets in string variables.

```
. use https://www.stata-press.com/data/r19/auto, clear
(1978 automobile data)

. describe make

Variable      Storage   Display    Value
    name         type    format    label      Variable label
-------------------------------------------------------------------------------
make            str18   %-18s                 Make and model
```

The string-variable storage types are `str1`, `str2`, …, `str2045`, and `strL`. For example, variable `make` is a `str18` variable. It can contain strings of up to 18 characters long. The strings are not all 18 characters long.

```
. list make in 1/2

     +--------------+
     | make         |
     |--------------|
  1. | AMC Concord  |
  2. | AMC Pacer    |
     +--------------+
```

str18 means that the variable cannot hold a string longer than 18 bytes, and even that is an unimportant detail, because Stata automatically promotes str# variables to be longer when required.

```
. replace make = "Mercedes Benz Gullwing" in 1
variable make was str18 now str22
(1 real change made)
```

Strings in Stata can also be stored in labels and notes that let you see information about your dataset. See **[U] 12.6 Dataset, variable, and value labels** and **[U] 12.7 Notes attached to data**. Strings in Stata programs can be stored in string scalars, macros, characteristics, and in stored results.

Stata provides a suite of string functions, such as strlen() and substr().

```
. generate len = strlen(make)

. generate str first5 = substr(make, 1,5)

. list make len first5 in 1/2

     ┌─────────────────────────────────────────┐
     │ make                     len    first5  │
     ├─────────────────────────────────────────┤
  1. │ Mercedes Benz Gullwing    22     Merce  │
  2. │ AMC Pacer                  9     AMC P  │
     └─────────────────────────────────────────┘
```

Many Stata commands can use string variables.

```
. generate str brand = word(make, 1)

. tabulate brand

      brand │      Freq.     Percent        Cum.
────────────┼───────────────────────────────────
        AMC │          2        2.70        2.70
       Audi │          2        2.70        5.41
        BMW │          1        1.35        6.76
      Buick │          7        9.46       16.22
       Cad. │          3        4.05       20.27
      Chev. │          6        8.11       28.38
     Datsun │          4        5.41       33.78
      Dodge │          4        5.41       39.19
       Fiat │          1        1.35       40.54
       Ford │          2        2.70       43.24
      Honda │          2        2.70       45.95
      Linc. │          3        4.05       50.00
      Mazda │          1        1.35       51.35
      Merc. │          6        8.11       59.46
   Mercedes │          1        1.35       60.81
      Olds  │          7        9.46       70.27
    Peugeot │          1        1.35       71.62
      Plym. │          5        6.76       78.38
      Pont. │          6        8.11       86.49
    Renault │          1        1.35       87.84
     Subaru │          1        1.35       89.19
     Toyota │          3        4.05       93.24
         VW │          4        5.41       98.65
      Volvo │          1        1.35      100.00
────────────┼───────────────────────────────────
      Total │         74      100.00
```

Beginning in Stata 14, text in Stata strings can include Unicode characters and is encoded as UTF-8. This means that you can use plain ASCII characters (also known as “lower ASCII” and stored as 0–127 on computers) like those shown above. You can also use the remaining Latin characters, as well as characters from the Chinese, Cyrillic, and Japanese alphabets, among others. However, if you have characters other than ASCII in your datasets, do-files, or ado-files, you may need to take special steps. See **[U] 12.4.2 Handling Unicode strings**.

12.4.2 Handling Unicode strings

If you do not have Unicode characters beyond the plain ASCII characters, you do not need to use any special steps to work with your data. In many cases, the same is true even if you do have other Unicode characters. While it is impossible to provide a rule for every situation, there are some general guidelines that you should be aware of.

The fundamental concept to understand is the difference between characters and bytes. Characters are what you see. For example, "a", "Z", and "@" are characters. Bytes are used to encode characters, which are stored on a computer.

For plain ASCII characters, there is a one-to-one mapping between the number of bytes and the number of characters. By contrast, UTF-8 encoded Unicode characters require two, three, or four bytes. For this reason, strings containing Unicode characters require string functions that recognize whole characters; see **[U] 12.4.2.1 Unicode string functions**. Some characters from older Stata files, known as extended ASCII characters, will not display correctly and can cause unexpected results. To avoid this, you must properly convert your older datasets and text files, such as do-files, if they contain extended ASCII. See **[U] 12.4.2.6 Advice for users of Stata 13 and earlier**.

If you do have characters in your data other than plain ASCII characters, or if you write commands for others to use, you should read the following sections.

12.4.2.1 Unicode string functions

Some of Stata's string functions exist in Unicode-aware versions so they can understand the string as a sequence of Unicode characters rather than as a sequence of bytes. At times, you will need to use one of these Unicode-aware functions to return accurate results. For example, suppose that our data on `make` included a car manufactured by Clénet Coachworks.

If we wanted to know the correct string length, we would use `ustrlen()`, not `strlen()`. The former will give you the answer you expect, 17, while the latter will return the number of bytes used to store that string, 18.

There are other Unicode-aware functions. For example, to change Unicode characters to uppercase, lowercase, or titlecase, use functions `ustrupper()`, `ustrlower()`, or `ustrtitle()`. If you want to see if there is a Unicode variant of the string function you want to use, check **[FN] String functions**.

Note that Unicode-aware functions are not required just because a variable contains UTF-8 characters beyond the plain ASCII range. For example, suppose that rather than wanting the string length, we wanted to replace "Mercedes" with "Merc.". We could use `subinstr()` instead of `usubinstr()` because neither "Mercedes" nor "Merc." contains UTF-8 characters.

Other Unicode-aware functions address the display columns. These functions are primarily of interest to programmers. See **[U] 12.4.2.2 Displaying Unicode characters**.

If you are in doubt, or if you are writing code to be used in a general way by others, you should use the Unicode-aware version of a string function, if it exists. The Unicode-aware functions generally have the same names as the regular string functions, but with "u" as a prefix. See **[FN] String functions**.

12.4.2.2 Displaying Unicode characters

Stata has a concept called a display column to ensure that the fixed-width output in Stata's Results and Viewer windows continues to align properly. Stata automatically displays each character in one or two display columns.

Most users, even users with UTF-8 characters beyond the ASCII range, will find that there is no distinction between the number of characters and the number of display columns because most characters are displayed in one column. Some wider characters, however, such as Chinese, Japanese, and Korean (CJK) characters, occupy two display columns.

You may occasionally wish to account for the number of display columns that a string occupies. Just as some Stata functions understand Unicode characters, some functions understand display columns. These functions are prefixed with "ud". For example, you can obtain the number of display columns for a string with `udstrlen(`*string*`)`. If you want to extract a subset of characters from the beginning of a string and make sure it fits within 10 display columns, use `udsubstr(`*string*`,1,10)`. See [FN] **String functions** for more information.

12.4.2.3 Encodings

An encoding is the way a computer stores a given string of text. ASCII and UTF-8, which is how Stata stores all text, are examples of encodings. Plain ASCII characters are stored as a single byte, each with a value between 0 and 127. "a", "Z", and "@" are all examples of plain ASCII characters, and their respective byte values are 97, 90, and 64.

The letter "á" is also a character. In UTF-8 encoding, that single character is stored as two bytes: 195 and 161. All Unicode characters beyond the plain ASCII range are stored as two or more bytes, and each of those bytes has a value between 128 and 255. Some characters in UTF-8 encoding take three or even four bytes to store.

Not every possible combination of bytes represents a valid Unicode character. Because two or more bytes are required to encode a Unicode character, any single byte between 128 and 255 is not a valid Unicode character. Invalid Unicode characters are most likely to occur if you have extended ASCII characters in a file from a previous version of Stata; see [U] **12.4.2.6 Advice for users of Stata 13 and earlier**.

If you have text in other encodings, including text in Stata files, you must convert it to UTF-8 for it to display properly and for some of Stata's string functions to work properly. To convert a file to UTF-8, you must know the original encoding. The most common encoding is Windows-1252. To obtain a list of other common encodings as well as a list of all possible encodings, see `unicode encoding list` and `unicode encoding alias` in [D] **unicode encoding**.

The `unicode analyze` and `unicode translate` commands help to convert text files and Stata datasets. See [D] **unicode translate** for more information. Also see [U] **12.4.2.6 Advice for users of Stata 13 and earlier**.

12.4.2.4 Locales in Unicode

A locale identifies a community with a certain set of rules for how their language should be written. A locale can be as general as a certain language, such as "en" for English, or it can be specific to a country or region, such as "en_US" for US English and "en_HK" for Hong Kong English.

Locales use tags to define how specific they are to language variants; these tags include language, script, country, variant, and keywords. Typically the language is required and the other tags are optional. In most cases, Stata uses only the language and country tags. For example, "en_US" specifies the language as English and the country as the USA.

Certain language-specific operations require a locale to be properly carried out. For example, in English, the uppercase version of "i" is "I". In Turkish, the uppercase version of "i" is an "İ" [that is, an "I" with a dot above it (Unicode character `\u0130`)]. To specify how to properly convert a letter to uppercase, you can specify the locale in the `ustrupper()` function, for example, `ustrupper("i", "en_US")`.

The following Stata functions are locale-dependent: `ustrupper()`, `ustrlower()`, `ustrtitle()`, `ustrword()`, `ustrwordcount()`, `ustrcompare()`, `ustrcompareex()`, `ustrsortkey()`, and `ustrsortkeyex()`.

If you do not explicitly specify a locale when using these functions, the current Stata `locale_functions` setting will be used. You can see the current setting by typing

```
. display c(locale_functions)
```

and

```
. unicode locale list
```

to see a list of supported locales. It is unlikely, however, that you will ever need to change the `set locale_functions` setting.

See [P] **set locale_functions** for more information about setting the locale, including information about how the default value is determined.

12.4.2.5 Sorting strings containing Unicode characters

This section deals with collation, sorting strings that contain Unicode characters, and the special rules that apply when you do. Many users will find that they can skip this section.

If you do not have Unicode characters beyond the plain ASCII range, you can skip this section. You can also skip this section if you are interested in using `sort` only so that you can use another command or prefix. For example, suppose you have the variable `id` that contains Unicode characters and you want to type

```
. statsby id: regress y x1 x2
```

If your aim is to group the coefficients by `id` only and the exact order of `id` does not matter, then the advice in this section does not apply to you. The usual `sort` command will be sufficient.

The steps described here also do not apply to commands that require the data to be sorted or grouped. For example, suppose that you wish to perform a one-to-one `merge` for two datasets using `id` as the key variable. You can just type

```
. merge 1:1 id using ...
```

Finally, you can skip this section if you do not want to apply language-specific rules to the Unicode characters in your data. For example, if you do not particularly care that "café" is sorted before or after "cafe", but only that the two words are distinguished, then this section is not for you.

For users who wish to sort or compare strings as a human might, there are four rules that you should keep in mind.

1. Sorting is locale-specific.
2. You must generate a sort key. You cannot sort by the variable itself.
3. There are multiple options for controlling the order of Unicode strings.
4. Concatenation is required to sort by *varlist*.

Rules 1 and 3 also apply to string comparisons. We explain each of these rules in more detail below. But first, it may be helpful to review how sorting works in general.

Stata's `sort` command and Stata's logical operators $>$ and $<$ order strings based on the byte values of the characters. For example, the byte value for "a" is 97 and the byte value for "A" is 65, so "a" $>$ "A". Similarly, the byte value for "Z" is 90, so "a" $>$ "Z". This means that words starting with "Z" come before "a", which might surprise you because, in an English dictionary, words starting with "Z" would certainly come after words starting with "a".

For example, suppose we have the following data:

```
. list mystr

     +-------+
     | mystr |
     |-------|
  1. | Quick |
  2. | quick |
  3. | brown |
  4. |   Fox |
  5. |  jump |
     +-------+
```

If we `sort` these data and then `list` them, we see

```
. sort mystr
. list

     +-------+
     | mystr |
     |-------|
  1. |   Fox |
  2. | Quick |
  3. | brown |
  4. |  jump |
  5. | quick |
     +-------+
```

This probably is not the order you would have placed these values in.

To sort the values of `mystr` in a more human fashion, you can use a Unicode tool, known as the Unicode collation algorithm (UCA), for comparing and sorting strings in a language-aware manner. Given knowledge of a locale and perhaps some optional instructions about whether to consider things like case and diacritical marks, the UCA can order Unicode strings as a human (or a dictionary) would.

Stata and Mata provide access to the UCA via the `ustrcompare()`, `ustrcompareex()`, and `ustrsortkey()`, `ustrsortkeyex()` functions. Stata also provides access via the `collatorlocale()` and `collatorversion()` functions.

See http://www.unicode.org/reports/tr10/ for the formal specification of the UCA.

Rule 1: Sorting is locale-dependent.

The ordering of strings in Unicode depends on the specified language and any optional tags and keywords that are specified with the locale.

For the `ustrcompare()` and `ustrsortkey()` functions, the default rules for ordering by language (and country, if specified) are used. You can use the current Stata `locale_functions` setting or specify a different locale with these each of these functions. See [U] **12.4.2.4 Locales in Unicode** for more information about locales, and see [D] **unicode collator** for information about locale-specific collation.

For advanced control of ordering, use the `ustrcompareex()` and `ustrsortkeyex()` functions. These functions allow you to specify a collation keyword, which is used for finer control for ordering, such as whether case-sensitivity and diacritical marks matter. For example, “pinyin” and “stroke” for the Chinese language produce different sort orders. A list of valid collation keywords and their meanings may be found http://unicode.org/repos/cldr/trunk/common/bcp47/collation.xml.

Rule 2: You must generate a sort key.

To appropriately sort your data with all the rules of the locale applied, you must generate a sort key. A sort key is a string created by the UCA that can be used to sort Unicode strings. You sort on the sort key rather than the Unicode string variable. The sort key is not a variable we would ever want to use for any purpose other than data management because it is not human-readable.

You can generate a sort key using either `ustrsortkey()` or `ustrsortkeyex()`. You then `sort` your data by the new variable. The following example illustrates the difference between `sort` and Unicode collation using the above functions:

```
. generate sortkey = ustrsortkey(mystr, "en")

. sort sortkey

. list mystr

     +-------+
     | mystr |
     |-------|
  1. | brown |
  2. |   Fox |
  3. |  jump |
  4. | quick |
  5. | Quick |
     +-------+
```

It is important to note that the Stata dataset is sorted by `sortkey` and not by `mystr`, even though `mystr` appears to be sorted correctly. Stata is aware of sorting only by `sortkey`. This means that if you need to perform an operation that relies on the sort order, such as `by`, you should use `sortkey` rather than `mystr`, such as

```
. by sortkey: ...
```

Also note that sort keys generated from one locale or one set of advanced options in `ustrsortkeyex()` are usually not compatible or comparable with sort keys generated from another locale or another set of options. For example, you should not compare the sort keys generated from the `"en"` locale with those generated from the `"fr"` locale.

❑ Technical note

The effective locale may be different from the requested locale. Thus, the sort keys obtained on a different machine, or even on a different user account on the same machine, may be different unless the locale is specified. You can retrieve the effective locale with the function `collatorlocale()` and then use that effective locale in future calls to the Unicode ordering functions.

❑

❑ Technical note

The Unicode standard is constantly adding more characters, and language rules are constantly changing, which means that sort keys produced by the current version of the UCA may not be compatible with sort keys of the same strings produced by future versions of the UCA.

You can use function `collatorversion()` to retrieve the current version of the collation routine and then store the result (for example, in a variable characteristic) with any saved sort keys if those keys are intended for future use.

If the current version is different from the saved sort key, then you should regenerate the sort key variables if you want them to be up-to-date with the new language rules or if you want to compare them with newly generated sort keys.

❑

Rule 3: There are multiple options for controlling the order of Unicode strings.

This may appear straightforward, but some finer points of the UCA could surprise you. Consider an example of string comparisons.

```
. display ustrcompare("café","cafe","fr")
1
```

Here we asked Stata to compare the string “café” with the string “cafe” using the French locale (`"fr"`). Stata reported `1`, which means that in this case “café” is considered to be greater than “cafe”. If we were sorting our data, this means “café” would be sorted after “cafe”.

Now consider

```
. display ustrcompare("café du monde","cafe new york","fr")
-1
```

It might surprise you that the result is `-1`, which means that in this case “café du monde” is considered to be less than “cafe new york”, even though we already established that ”café” is greater than “cafe”.

The reason is that the difference between “d” and “n” in the second word of each string is considered by the UCA to be a primary difference, whereas the difference between “é” and “e” in the first word of each string is a diacritical mark which is considered to be a secondary difference. The primary difference outweighs the secondary difference even though it occurs later in the string.

The default behavior of `ustrcompare()` and `ustrsortkey()` should be sufficient for most comparison and sorting needs. For advanced control over how Unicode strings are ordered, including whether the ordering should be based on differences from primary to quaternary, use `ustrcompareex()` and `ustrsorkeyex()`. See [FN] **String functions**.

Rule 4: Concatenation is required to sort by a varlist.

An important implication of Rule 3 arises when creating sort keys for Unicode strings. Ordinarily, if you want to sort on two string variables, you can simply type

```
. sort string1 string2
```

However, to take full advantage of the UCA while sorting two or more strings, you should first concatenate them and then sort the result.

```
. generate string3 = string1 + string2
. generate sortkey = ustrsortkey(sting3, "fr")
. sort sortkey
```

If you do not do this, then primary differences that might arise in `string2` will not override any secondary differences in `string1`.

12.4.2.6 Advice for users of Stata 13 and earlier

In this section, we discuss how to use your older Stata files in modern Stata and also points you should consider when sharing your modern Stata files with users of Stata 13 and earlier.

In Stata 13 and earlier, Unicode characters were not supported. If you have only plain ASCII characters in your datasets, do-files, and ado-files, then you do not need to take any special steps to continue using these files with modern Stata. You can use `saveold` to share your dataset with users of older versions of Stata. Your do-files and ado-files can be shared directly.

If files you used with Stata 13 or earlier contain strings with extended ASCII characters, you should convert those strings to Unicode UTF-8 encoding so they will work properly with modern Stata. The `unicode analyze` command will check your files to see if they need conversion, and if so, the `unicode translate` command will convert them to UTF-8 encoding. See [D] **unicode translate**. To convert a single variable, use `ustrfrom()`.

If you have Unicode characters in your dataset and you wish to share it with a user of Stata 13 or earlier, be aware that while they can load a dataset created with the `saveold` command, their copy of Stata is not Unicode-aware and will not display Unicode characters properly. Before you use `saveold`, you can convert your string variables from the UTF-8 encoding to an extended ASCII encoding by using `ustrto()`. We recommend that you `generate` a new variable when using `ustrfrom()` or `ustrto()` so that you can review the results and make sure you are satisfied before you `replace` your existing variable. `ustrfrom()` and `ustrto()` may also be used with Mata string matrices.

12.4.3 Strings containing identifying data

String variables often contain identifying information, such as the patient's name or the name of the city or state. Such strings are typically listed but are not used directly in statistical analysis, although the data might be sorted on the string or datasets might be merged on the basis of one or more string variables.

12.4.4 Strings containing categorical data

Strings sometimes contain information to be used directly in analysis, such as the patient's sex, which might be coded "male" or "female". Stata shows a decided preference for such information to be numerically encoded and stored in numeric variables. Stata's statistical routines treat string variables as if every observation records a numeric missing value. Stata provides two commands for converting string variables into numeric codes and back again: `encode` and `decode`. See [U] **24.2 Categorical string variables** and [U] **11.4.3 Factor variables**.