GETTING STARTED WITH STATA
FOR WINDOWS®
RELEASE 17

The suggested citation for this software is

StataCorp. 2021. *Stata: Release 17*. Statistical Software. College Station, TX: StataCorp LLC.

Contents

Cross-referencing the documentation

When reading this manual, you will find references to other Stata manuals, for example, [U] 27 Overview of Stata estimation commands; [R] regress; and [D] reshape. The first example is a reference to chapter 27, *Overview of Stata estimation commands*, in the *User's Guide*; the second is a reference to the regress entry in the *Base Reference Manual*; and the third is a reference to the reshape entry in the *Data Management Reference Manual*.

All the manuals in the Stata Documentation have a shorthand notation:

[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*
[BAYES]	*Stata Bayesian Analysis 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*
[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*
[TE]	*Stata Treatment-Effects Reference Manual:* *Potential Outcomes/Counterfactual Outcomes*
[I]	*Stata Index*
[M]	*Mata Reference Manual*

About this manual

This manual discusses **Stata for Windows**®. Stata for Mac® users should see *Getting Started with Stata for Mac*; Stata for Unix® users should see *Getting Started with Stata for Unix*. This manual is intended both for people who are completely new to Stata and for experienced Stata users new to Stata for Windows. Previous Stata users will also find it helpful as a tutorial on some new features in Stata for Windows.

Following the numbered chapters are three appendixes with information specific to Stata for Windows.

We provide several types of technical support to registered Stata users. [GSW] **4 Getting help** describes the resources available to help you learn about Stata's commands and features. One of these resources is the Stata website (https://www.stata.com), where you will find answers to frequently asked questions (FAQs) as well as other useful information. If you still have questions after looking at the Stata website and the other resources described in [GSW] **19 Updating and extending Stata—Internet functionality**, you can contact us as described in [U] **3.8 Technical support**.

Using this manual

The new user will get the most out of this book by treating it as an exercise book, working through each example at the computer. The material builds, so material from earlier chapters will often be used in later chapters. Bear in mind that Stata is a rich and deep statistical package—just as statistics itself is rich and deep. The time spent working the examples will be repaid with dividends when doing true statistical analyses.

The experienced user may still have something to learn from this manual despite its name. We suggest looking through the chapters to see if there is anything new or forgotten.

1 Introducing Stata—sample session

Introducing Stata

This chapter will run through a sample work session, introducing you to a few of the basic tasks that can be done in Stata, such as opening a dataset, investigating the contents of the dataset, using some descriptive statistics, making some graphs, and doing a simple regression analysis. As you would expect, we will only brush the surface of many of these topics. This approach should give you a sample of what Stata can do and how Stata works. There will be brief explanations along the way, with references to chapters later in this book as well as to the system help and other Stata manuals. We will run through the session by using both menus and dialogs and Stata's commands so that you can become familiar with them both. If you see that your menus and dialogs are not in English, we recommend that you (temporarily) change the locale used by Stata to English, so that you can work along with the examples. See [P] **set locale_ui** for how to do this.

Take a seat at your computer, put on some good music, and work along with the book.

Sample session

The dataset that we will use for this session is a set of data about vintage 1978 automobiles sold in the United States.

To follow along by pointing and clicking, note that the menu items are given by **Menu > Menu item > Submenu item > etc**. To follow along by using the Command window, type the commands that follow a dot (.) in the boxed listings below into the small window labeled **Command**. When there is something to note about the structure of a command, it will be pointed out as a "Syntax note".

Start by loading the automobile dataset, which is included with Stata. Use the menus to do this:
1. Select **File > Example datasets...**.
2. Click on Example datasets installed with Stata.
3. Click on use for auto.dta.

The result of this command is fourfold:
- The following output appears in the large Results window:

```
. sysuse auto
(1978 automobile data)
```

  The output consists of a command and its result. The command, sysuse auto.dta, is bold and follows the dot (.). The result, (1978 automobile data), is in the standard face here and is a brief description of the dataset.
  Note: If a command intrigues you, you can type help *commandname* in the Command window to find help. If you want to explore at any time, **Help > Search...** can be informative.
- The same command, sysuse auto.dta, appears in the tall History window to the left. The History window keeps track of all commands Stata has run, successful and unsuccessful. The commands can then easily be rerun. See [GSW] **2 The Stata user interface** for more information.

- A series of variables appears in the small Variables window to the upper right.
- Some information about `make`, the first variable in the dataset, appears in the small Properties window to the lower right.

You could have opened the dataset by typing `sysuse auto` in the Command window and pressing *Enter*. Try this now. `sysuse` is a command that loads (uses) example (system) datasets. As you will see during this session, Stata commands are often simple enough that it is faster to use them directly. This will be especially true once you become familiar with the commands you use the most in your daily use of Stata.

Syntax note: In the above example, `sysuse` is the Stata command, whereas `auto` is the name of a Stata data file.

Simple data management

We can get a quick glimpse at the data by browsing them in the **Data Editor**. This can be done by clicking on the **Data Editor (Browse)** button, [icon], or by selecting **Data > Data Editor > Data Editor (Browse)** from the menus or by typing the command `browse`.

No command is issued when clicking on either Data Editor button because opening the Data Editor has no effect on the dataset or any possible analysis.

When the Data Editor window opens, you can see that Stata regards the data as one rectangular table. This is true for all Stata datasets. The columns represent *variables*, whereas the rows represent *observations*. The variables have somewhat descriptive names, whereas the observations are numbered.

The data are displayed in multiple colors—at first glance, it appears that the variables listed in black are numeric, whereas those that are in colors are text. This is worth investigating. Click on a cell under the `make` variable: the input box at the top displays the make of the car. Scroll to the right until you see the `foreign` variable. Click on one of its cells. Although the cell may display "Domestic", the input box displays a 0. This shows that Stata can store categorical data as numbers but display human-readable text. This is done by what Stata calls *value labels*. Finally, under the `rep78` variable, which looks to be numeric, there are some cells containing just a dot (`.`). The dots correspond to missing values.

Looking at the data in this fashion, though comfortable, lends little information about the dataset. It would be useful for us to get more details about what the data are and how the data are stored. Close the Data Editor by clicking on its close button.

We can see the structure of the dataset by *describing* its contents. This can be done either by going to **Data > Describe data > Describe data in memory or in a file** in the menus and clicking on **OK** or by typing `describe` in the Command window and pressing *Enter*. Regardless of which method you choose, you will get the same result:

```
. describe

Contains data from C:\Program Files\Stata17/ado/base/a/auto.dta
 Observations:            74                  1978 automobile data
    Variables:            12                  13 Apr 2020 17:45
                                              (_dta has notes)
────────────────────────────────────────────────────────────────────────
Variable      Storage   Display    Value
    name         type    format    label      Variable label
────────────────────────────────────────────────────────────────────────
make            str18    %-18s                 Make and model
price           int      %8.0gc                Price
mpg             int      %8.0g                 Mileage (mpg)
rep78           int      %8.0g                 Repair record 1978
headroom        float    %6.1f                 Headroom (in.)
trunk           int      %8.0g                 Trunk space (cu. ft.)
weight          int      %8.0gc                Weight (lbs.)
length          int      %8.0g                 Length (in.)
turn            int      %8.0g                 Turn circle (ft.)
displacement    int      %8.0g                 Displacement (cu. in.)
gear_ratio      float    %6.2f                 Gear ratio
foreign         byte     %8.0g      origin     Car origin
────────────────────────────────────────────────────────────────────────
Sorted by: foreign
```

At the top of the listing, some information is given about the dataset, such as where it is stored on disk and when the dataset was last saved. The bold `1978 automobile data` is the short description that appeared when the dataset was opened and is referred to as a *data label* by Stata. The phrase `_dta has notes` informs us that there are notes attached to the dataset. We can see what notes there are by typing `notes` in the Command window:

```
. notes

_dta:
  1.  From Consumer Reports with permission
```

Here we see a short note about the source of the data.

Looking back at the listing from `describe`, we can see that Stata keeps track of more than just the raw data. Each variable has the following:

- A *variable name*, which is what you call the variable when communicating with Stata. Variable names are one type of Stata name. See [U] **11.3 Naming conventions**.
- A *storage type*, which is the way Stata stores its data. For our purposes, it is enough to know that types like, say, `str#` are *string*, or text, variables, whereas all others in this dataset are numeric. While there are none in this dataset, Stata also allows arbitrarily long strings, or `strLs`. `strLs` can also contain binary information. See [U] **12.4 Strings**.
- A *display format*, which controls how Stata displays the data in tables. See [U] **12.5 Formats: Controlling how data are displayed**.
- A *value label* (possibly). This is the mechanism that allows Stata to store numerical data while displaying text. See [GSW] **9 Labeling data** and [U] **12.6.3 Value labels**.
- A *variable label*, which is what you call the variable when communicating with other people. Stata uses the variable label when making tables, as we will see.

A dataset is far more than simply the data it contains. It is also information that makes the data usable by someone other than the original creator.

Although describing the data tells us something about the structure of the data, it says little about the data themselves. The data can be summarized by clicking on **Statistics > Summaries, tables, and tests > Summary and descriptive statistics > Summary statistics** and clicking on the **OK** button. You could also type `summarize` in the Command window and press *Enter*. The result is a table containing summary statistics about all the variables in the dataset:

```
. summarize
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
        make |          0
       price |         74    6165.257    2949.496       3291      15906
         mpg |         74     21.2973    5.785503         12         41
       rep78 |         69    3.405797    .9899323          1          5
     headroom |        74    2.993243    .8459948        1.5          5
-------------+---------------------------------------------------------
       trunk |         74    13.75676    4.277404          5         23
      weight |         74    3019.459    777.1936       1760       4840
      length |         74    187.9324    22.26634        142        233
        turn |         74    39.64865    4.399354         31         51
displacement |         74    197.2973    91.83722         79        425
-------------+---------------------------------------------------------
   gear_ratio |        74    3.014865    .4562871       2.19       3.89
     foreign |         74    .2972973    .4601885          0          1
```

From this simple summary, we can learn a bit about the data. First of all, the prices are nothing like today's car prices—of course, these cars are now antiques. We can see that the gas mileages are not particularly good. Automobile aficionados can get a feel for other esoteric characteristics.

There are two other important items here:

- The `make` variable is listed as having no observations. It really has no numerical observations because it is a string (text) variable.
- The `rep78` variable has five fewer observations than the other numerical variables. This implies that `rep78` has five missing values.

Although we could use the `summarize` and `describe` commands to get a bird's eye view of the dataset, Stata has a command that gives a good in-depth description of the structure, contents, and

values of the variables: the `codebook` command. Either type `codebook` in the Command window and press *Enter* or navigate the menus to **Data > Describe data > Describe data contents (codebook)** and click on **OK**. Look over the output to see that much can be learned from this simple command. You can scroll back in the Results window to see earlier results, if need be. We will focus on the output for `make`, `rep78`, and `foreign`.

To start our investigation, we would like to run the `codebook` command on just one variable, say, `make`. We can do this, as usual, with menus or the command line. To get the `codebook` output for `make` with the menus, start by navigating to **Data > Describe data > Describe data contents (codebook)**. When the dialog appears, there are multiple ways to tell Stata to consider only the `make` variable:

- We could type `make` into the *Variables* field.
- The *Variables* field is a combo-box control that accepts variable names. Clicking on the drop triangle to the right of the *Variables* field displays a list of the variables from the current dataset. Selecting a variable from the list will, in this case, enter the variable name into the edit field.

A much easier solution is to type `codebook make` in the Command window and then press *Enter*. The result is informative:

```
. codebook make

make                                                      Make and model
─────────────────────────────────────────────────────────────────────────
                Type: String (str18), but longest is str17

       Unique values: 74                        Missing "": 0/74

            Examples: "Cad. Deville"
                      "Dodge Magnum"
                      "Merc. XR-7"
                      "Pont. Catalina"

             Warning: Variable has embedded blanks.
```

The first line of the output tells us the variable name (`make`) and the variable label (`Make and model`). The variable is stored as a string (which is another way of saying "text") with a maximum length of 18 characters, though a size of only 17 characters would be enough. All the values are unique, so if need be, `make` could be used as an identifier for the observations—something that is often useful when putting together data from multiple sources or when trying to weed out errors from the dataset. There are no missing values, but there are blanks within the makes. This latter fact could be useful if we were expecting `make` to be a one-word string variable.

Syntax note: Telling the `codebook` command to run on the `make` variable is an example of using a *varlist* in Stata's syntax.

Looking at the `foreign` variable can teach us about value labels. We would like to look at the codebook output for this variable, and on the basis of our latest experience, it would be easy to type `codebook foreign` into the Command window (from here on, we will not explicitly say to press the *Enter* key) to get the following output:

```
. codebook foreign
```

```
foreign                                                                Car origin
```

```
                  Type: Numeric (byte)
                 Label: origin

                 Range: [0,1]                      Units: 1
         Unique values: 2                      Missing .: 0/74

            Tabulation: Freq.    Numeric  Label
                           52          0  Domestic
                           22          1  Foreign
```

We can glean that `foreign` is an indicator variable because its only values are 0 and 1. The variable has a value label that displays `Domestic` instead of 0 and `Foreign` instead of 1. There are two advantages of storing the data in this form:

- Storing the variable as a byte takes less memory because each observation uses 1 byte instead of the 8 bytes needed to store "Domestic". This is important in large datasets. See [U] **12.2.2 Numeric storage types**.
- As an indicator variable, it is easy to incorporate into statistical models. See [U] **26 Working with categorical data and factor variables**.

Finally, we can learn a little about a poorly labeled variable with missing values by looking at the `rep78` variable. Typing `codebook rep78` into the Command window yields

```
. codebook rep78
```

```
rep78                                                          Repair record 1978
```

```
                  Type: Numeric (int)

                 Range: [1,5]                      Units: 1
         Unique values: 5                      Missing .: 5/74

            Tabulation: Freq.  Value
                            2  1
                            8  2
                           30  3
                           18  4
                           11  5
                            5  .
```

`rep78` appears to be a categorical variable, but because of lack of documentation, we do not know what the numbers mean. (To see how we would label the values, see *Changing data* in [GSW] **6 Using the Data Editor** and see [GSW] **9 Labeling data**.) This variable has five missing values, meaning that there are five observations for which the repair record is not recorded. We could use the Data Editor to investigate these five observations, but we will do this by using the Command window only because doing so is much simpler.

The command equivalent to clicking on the **Data Editor (Browse)** button is `browse`. We would like to browse only those observations for which `rep78` is missing, so we could type

```
. browse if missing(rep78)
```

	make	price	mpg	rep78	headroom
3	AMC Spirit	3,799	22	.	3.0
7	Buick Opel	4,453	26	.	3.0
45	Plym. Sapporo	6,486	26	.	1.5
51	Pont. Phoenix	4,424	19	.	3.5
64	Peugeot 604	12,990	14	.	3.5

From this, we see that the `.` entries are indeed missing values. The `.` is the default numerical missing value; Stata also allows `.a, ..., .z` as user missing values, but we do not have any in our dataset. See [U] **12.2.1 Missing values**. Close the Data Editor after you are satisfied with this statement.

Syntax note: Using the `if` qualifier above is what allowed us to look at a subset of the observations.

Looking through the data lends no clues about why these particular data are missing. We decide to check the source of the data to see if the missing values were originally missing or if they were omitted in error. Listing the makes of the cars whose repair records are missing will be all we need because we saw earlier that the values of `make` are unique. This can be done with the menus and a dialog:

1. Select **Data > Describe data > List data**.
2. Click on the drop triangle to the right of the *Variables* field to show the variable names.
3. Click on `make` to enter it into the *Variables* field.
4. Click on the **by/if/in** tab in the dialog.
5. Type `missing(rep78)` into the *If: (expression)* box.
6. Click on **Submit**. Stata executes the proper command but the dialog remains open. **Submit** is useful when experimenting, exploring, or building complex commands. We will primarily use **Submit** in the examples. You may click on **OK** in its place if you like, and it will close the dialog box.

The same ends could be achieved by typing `list make if missing(rep78)` in the Command window. The latter is easier once you know that the command `list` is used for listing observations. In any case, here is the output:

```
. list make if missing(rep78)

        ┌───────────────┐
        │ make          │
        ├───────────────┤
  3.    │ AMC Spirit    │
  7.    │ Buick Opel    │
 45.    │ Plym. Sapporo │
 51.    │ Pont. Phoenix │
 64.    │ Peugeot 604   │
        └───────────────┘
```

At this point, we should find the original reference to see if the data were truly missing or if they could be resurrected. See [GSW] **10 Listing data and basic command syntax** for more information about all that can be done with the list command.

Syntax note: This command uses two new concepts for Stata commands—the if qualifier and the missing() function. The if qualifier restricts the observations on which the command runs to only those observations for which the expression is true. See [U] **11.1.3 if exp**. The missing() function tests each observation to see if it contains a missing value. See [FN] **Programming functions**.

Now that we have a good idea about the underlying dataset, we can investigate the data themselves.

Descriptive statistics

We saw above that the summarize command gave brief summary statistics about all the variables. Suppose now that we became interested in the prices while summarizing the data because they seemed fantastically low (it was 1978, after all). To get an in-depth look at the price variable, we can use the menus and a dialog:

1. Select **Statistics > Summaries, tables, and tests > Summary and descriptive statistics > Summary statistics**.
2. Enter or select price in the *Variables* field.
3. Select *Display additional statistics*.
4. Click on **Submit**.

Syntax note: As can be seen from the Results window, typing summarize price, detail will get the same result. The portion after the comma contains *options* for Stata commands; hence, detail is an example of an option.

```
. summarize price, detail

                            Price

      Percentiles      Smallest
 1%         3291          3291
 5%         3748          3299
10%         3895          3667       Obs                      74
25%         4195          3748       Sum of wgt.              74

50%       5006.5                     Mean              6165.257
                        Largest      Std. dev.         2949.496
75%         6342         13466
90%        11385         13594       Variance           8699526
95%        13466         14500       Skewness          1.653434
99%        15906         15906       Kurtosis          4.819188
```

From the output, we can see that the median price of the cars in the dataset is only $5,006.50! We can also see that the four most expensive cars are all priced between $13,400 and $16,000. If we wished to browse the most expensive cars (and gain some experience with features of the Data Editor), we could start by clicking on the **Data Editor (Browse)** button, 🔲 .

Once the Data Editor is open, we can click on the **Filter observations** button, ▼ , to bring up the *Filter observations* dialog. We can look at the expensive cars by putting `price > 13000` in the *Filter by expression* field:

Pressing the **Apply filter** button filters the data, and we can see that the expensive cars are two Cadillacs and two Lincolns, which were not designed for gas mileage:

Data Editor (Browse) - [auto]

File   Edit   View   Data   Tools

make[12]   Cad. Eldorado

	make	price	mpg	rep78	headroom
12	Cad. Eldorado	14,500	14	2	3.5
13	Cad. Seville	15,906	21	3	3.0
27	Linc. Mark V	13,594	12	3	2.5
28	Linc. Versailles	13,466	14	3	3.5

Variables

Filter variables here

✓	Name	Label
✓	make	Make and model

Variables   Snapshots

Properties

▲ Variables

Name   make

Ready   Length: 18   Vars: 12 Order: Dataset   Obs: 4 of 74   Filter: On   Mode: Browse   CAP   NUM

We now decide to turn our attention to foreign cars and repairs because as we glanced through the data, it appeared that the foreign cars had better repair records. (We do not know exactly what the categories 1, 2, 3, 4, and 5 mean, but we know the Chevy Monza was known for breaking down.) Let's start by looking at the proportion of foreign cars in the dataset along with the proportion of cars with each type of repair record. We can do this with one-way tables. The table for `foreign` cars can be done with menus and a dialog starting with **Statistics > Summaries, tables, and tests > Frequency tables > One-way table** and then choosing the variable `foreign` in the *Categorical variable* field. Clicking on **Submit** yields

```
. tabulate foreign

 Car origin |      Freq.     Percent        Cum.
------------+-----------------------------------
   Domestic |         52       70.27       70.27
    Foreign |         22       29.73      100.00
------------+-----------------------------------
      Total |         74      100.00
```

We see that roughly 70% of the cars in the dataset are domestic, whereas 30% are foreign. The value labels are used to make the table so that the output is nicely readable.

Syntax note: We also see that this one-way table could be made by using the `tabulate` command together with one variable, `foreign`. Making a one-way table for the repair records is simple—it will be simpler if done with the Command window. Typing `tabulate rep78` yields

```
. tabulate rep78

     Repair |
record 1978 |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |          2        2.90        2.90
          2 |          8       11.59       14.49
          3 |         30       43.48       57.97
          4 |         18       26.09       84.06
          5 |         11       15.94      100.00
------------+-----------------------------------
      Total |         69      100.00
```

We can see that most cars have repair records of 3 and above, though the lack of value labels makes us unsure what a "3" means. Take our word for it that 1 means a poor repair record and 5 means a good repair record. The five missing values are indirectly evident because the total number of observations listed is 69 rather than 74.

These two one-way tables do not help us compare the repair records of foreign and domestic cars. A two-way table would help greatly, which we can get by using the menus and a dialog:

1. Select **Statistics > Summaries, tables, and tests > Frequency tables > Two-way table with measures of association**.
2. Choose `rep78` as the *Row variable*.
3. Choose `foreign` as the *Column variable*.
4. It would be nice to have the percentages within the `foreign` variable, so check the *Within-row relative frequencies* checkbox.
5. Click on **Submit**.

Here is the resulting output:

```
. tabulate rep78 foreign, row

  ┌─────────────────┐
  │ Key             │
  ├─────────────────┤
  │     frequency   │
  │ row percentage  │
  └─────────────────┘

  Repair  │
  record  │      Car origin
    1978  │  Domestic    Foreign  │      Total
──────────┼──────────────────────┼───────────
       1  │         2          0  │          2
          │    100.00       0.00  │     100.00
──────────┼──────────────────────┼───────────
       2  │         8          0  │          8
          │    100.00       0.00  │     100.00
──────────┼──────────────────────┼───────────
       3  │        27          3  │         30
          │     90.00      10.00  │     100.00
──────────┼──────────────────────┼───────────
       4  │         9          9  │         18
          │     50.00      50.00  │     100.00
──────────┼──────────────────────┼───────────
       5  │         2          9  │         11
          │     18.18      81.82  │     100.00
──────────┼──────────────────────┼───────────
   Total  │        48         21  │         69
          │     69.57      30.43  │     100.00
```

The output indicates that foreign cars are generally much better than domestic cars when it comes to repairs. If you like, you could repeat the previous dialog and try some of the hypothesis tests available from the dialog. We will abstain.

Syntax note: We see that typing the command `tabulate rep78 foreign, row` would have given us the same table. Thus using `tabulate` with two variables yields a two-way table. It makes sense that `row` is an option—we went out of our way to check it in the dialog. Using the `row` option allows us to change the behavior of the `tabulate` command from its default.

Continuing our exploratory tour of the data, we would like to compare gas mileages between foreign and domestic cars, starting by looking at the summary statistics for each group by itself. A

direct way to do this would be to use `if` qualifiers to summarize `mpg` for each of the two values of `foreign` separately:

```
. summarize mpg if foreign==0
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
         mpg |         52    19.82692    4.743297         12         34
. summarize mpg if foreign==1
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
         mpg |         22    24.77273    6.611187         14         41
```

It appears that foreign cars get somewhat better gas mileage—we will test this soon.

Syntax note: We needed to use a double equal sign (`==`) for testing equality. The double equal sign could be familiar to you if you have programmed before. If it is unfamiliar, be aware that it is a common source of errors when initially using Stata. Thinking of equality as "exactly equal" can cut down on typing errors.

There are two other methods that we could have used to produce these summary statistics. These methods are worth knowing because they are less error-prone. The first method duplicates the concept of what we just did by exploiting Stata's ability to run a command on each of a series of nonoverlapping subsets of the dataset. To use the menus and a dialog, do the following:

1. Select **Statistics > Summaries, tables, and tests > Summary and descriptive statistics > Summary statistics** and click on the **Reset** button, ⟳.
2. Select `mpg` in the *Variables* field.
3. Select the *Standard display* option (if it is not already selected).
4. Click on the **by/if/in** tab.
5. Check the *Repeat command by groups* checkbox.
6. Select or type `foreign` in the *Variables that define groups* field.
7. **Submit** the command.

You can see that the results match those from above. They have a better appearance than the two commands above because the value labels `Domestic` and `Foreign` are used rather than the numerical values. The method is more appealing because the results were produced without needing to know the possible values of the grouping variable ahead of time.

```
. by foreign, sort: summarize mpg

-> foreign = Domestic
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
         mpg |         52    19.82692    4.743297         12         34

-> foreign = Foreign
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
         mpg |         22    24.77273    6.611187         14         41
```

Syntax note: There is something different about the equivalent command that appears above: it contains a *prefix command* called a `by` prefix. The `by` prefix has its own option, namely, `sort`, to ensure that like members are adjacent to each other before being summarized. The `by` prefix command is

important for understanding data manipulation and working with subpopulations within Stata. Make good note of this example, and consult [U] **11.1.2 by varlist:** and [U] **13.7 Explicit subscripting** for more information. Stata has other prefix commands for specialized treatment of commands, as explained in [U] **11.1.10 Prefix commands**.

The third method for tabulating the differences in gas mileage across the cars' origins involves thinking about the structure of desired output. We need a one-way table of automobile types (foreign versus domestic) within which we see information about gas mileages. Looking through the menus yields the menu item **Statistics > Summaries, tables, and tests > Other tables > Table of means, std. dev., and frequencies**. Selecting this, entering `foreign` for *Variable 1* and `mpg` for the *Summarize variable*, and submitting the command yields a nice table:

```
. tabulate foreign, summarize(mpg)
                  Summary of Mileage (mpg)
   Car origin         Mean    Std. dev.         Freq.

     Domestic    19.826923    4.7432972            52
      Foreign    24.772727    6.6111869            22

        Total    21.297297    5.7855032            74
```

The equivalent command is evidently `tabulate foreign, summarize(mpg)`.

Syntax note: This is a one-way table, so `tabulate` uses one variable. The variable being summarized is passed to the `tabulate` command with an option. Though we will not do it here, the `summarize()` option can also be used with two-way tables.

A simple hypothesis test

We would like to run a hypothesis test for the difference in the mean gas mileages. Under the menus, **Statistics > Summaries, tables, and tests > Classical tests of hypotheses > t test (mean-comparison test)** leads to the proper dialog. Select the *Two-sample using groups* radio button, enter `mpg` for the *Variable name* and `foreign` for the *Group variable name*, and **Submit** the dialog. The results are

```
. ttest mpg, by(foreign)
Two-sample t test with equal variances

    Group        Obs        Mean     Std. err.    Std. dev.    [95% conf. interval]

 Domestic         52    19.82692     .657777     4.743297     18.50638    21.14747
  Foreign         22    24.77273     1.40951     6.611187     21.84149    27.70396

 Combined         74     21.2973    .6725511     5.785503      19.9569    22.63769

     diff               -4.945804    1.362162                 -7.661225   -2.230384

     diff = mean(Domestic) - mean(Foreign)                           t =  -3.6308
Ho: diff = 0                                   Degrees of freedom =        72

   Ha: diff < 0                  Ha: diff != 0                  Ha: diff > 0
Pr(T < t) = 0.0003        Pr(|T| > |t|) = 0.0005        Pr(T > t) = 0.9997
```

From this, we could conclude that the mean gas mileage for foreign cars is different from that of domestic cars (though we really ought to have wanted to test this before snooping through the data).

We can also conclude that the command, `ttest mpg, by(foreign)`, is easy enough to remember. Feel free to experiment with unequal variances, various approximations to the number of degrees of freedom, and the like.

Syntax note: The `by()` option used here is not the same as the `by` prefix command used earlier. Although it has a similar conceptual meaning, its usage is different because it is a particular option for the `ttest` command.

Descriptive statistics—correlation matrices

We now change our focus from exploring categorical relationships to exploring numerical relationships: we would like to know if there is a correlation between miles per gallon and weight. We select **Statistics > Summaries, tables, and tests > Summary and descriptive statistics > Correlations and covariances** in the menus. Entering `mpg` and `weight`, either by clicking or by typing, and then submitting the command yields

```
. correlate mpg weight
(obs=74)
                   mpg    weight

       mpg       1.0000
    weight      -0.8072    1.0000
```

The equivalent command for this is natural: `correlate mpg weight`. There is a negative correlation, which is not surprising because heavier cars should be harder to push about.

We could see how the correlation compares for foreign and domestic cars by using our knowledge of the `by` prefix. We can reuse the *correlate* dialog or use the menus as before if the dialog is closed. Click on the **by/if/in** tab, check the *Repeat command by groups* checkbox, and enter the `foreign` variable to define the groups. As done on page 12, a simple `by foreign, sort:` prefix in front of our previous command would work, too:

```
. by foreign, sort: correlate mpg weight

-> foreign = Domestic
(obs=52)
                   mpg    weight

       mpg       1.0000
    weight      -0.8759    1.0000


-> foreign = Foreign
(obs=22)
                   mpg    weight

       mpg       1.0000
    weight      -0.6829    1.0000
```

We see from this that the correlation is not as strong among the foreign cars.

Syntax note: Although we used the `correlate` command to look at the correlation of two variables, Stata can make correlation matrices for an arbitrary number of variables:

```
. correlate mpg weight length turn displacement
(obs=74)

                     mpg    weight    length      turn  displa~t

        mpg       1.0000
     weight      -0.8072    1.0000
     length      -0.7958    0.9460    1.0000
       turn      -0.7192    0.8574    0.8643    1.0000
displacement     -0.7056    0.8949    0.8351    0.7768    1.0000
```

This can be useful, for example, when investigating collinearity among predictor variables.

Graphing data

We have found several things in our investigations so far: We know that the average MPG of domestic and foreign cars differs. We have learned that domestic and foreign cars differ in other ways as well, such as in frequency-of-repair record. We found a negative correlation between MPG and weight—as we would expect—but the correlation appears stronger for domestic cars.

We would now like to examine, with an eye toward modeling, the relationship between MPG and weight, starting with a graph. We can start with a scatterplot of `mpg` against `weight`. The command for this is simple: `scatter mpg weight`. Using the menus requires a few steps because the graphs in Stata may be customized heavily.

1. Select **Graphics > Twoway graph (scatter, line, etc.)**.
2. Click on the **Create...** button.
3. Select the *Basic plots* radio button (if it is not already selected).
4. Select *Scatter* as the basic plot type (if it is not already selected).
5. Select mpg as the *Y variable* and weight as the *X variable.*
6. Click on the **Submit** button.

The Results window shows the command that was issued from the menu:

```
. twoway (scatter mpg weight)
```

The command issued when the dialog was submitted is a bit more complex than the command suggested above. There is good reason for this: the more complex structure allows combining and overlaying graphs, as we will soon see. In any case, the graph that appears is
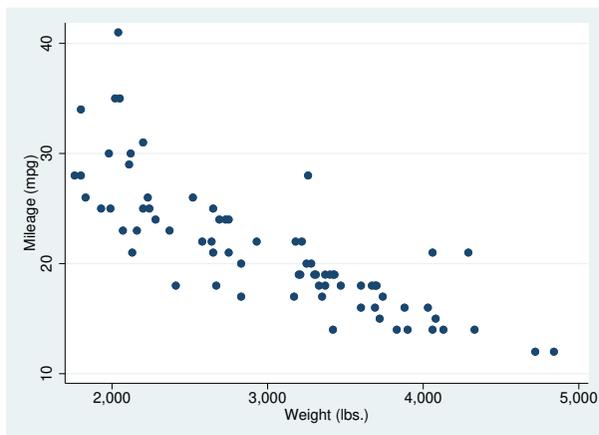
We see the negative correlation in the graph, though the relationship appears to be nonlinear.

Note: When you draw a graph, the Graph window appears, probably covering up your Results window. Click on the main Stata window to get the Results window back on top. Want to see the graph again? Click on the **Graph** button, [icon] . See *The Graph button* in [GSW] **14 Graphing data** for more information about the **Graph** button.

We would now like to see how the different correlations for foreign and domestic cars are manifested in scatterplots. It would be nice to see a scatterplot for each type of car, along with a scatterplot for all the data.

Syntax note: Because we are looking at subgroups, this looks as if it is a job for the by prefix. Let's see if this is what we really should use.

Start as before:
1. Select **Graphics > Twoway graph (scatter, line, etc.)** from the menus.
2. If the *Plot 1* dialog is still visible, click on the **Accept** button and skip to step 4.
3. Go through the process on the previous page to create the graph.
4. Click on the **By** tab of the *twoway - Twoway graphs* dialog.
5. Check the *Draw subgraphs for unique values of variables* checkbox.
6. Enter foreign in the *Variables* field.
7. Check the *Add a graph with totals* checkbox.
8. Click on the **Submit** button.

The command and the associated graph are

```
. twoway (scatter mpg weight), by(foreign, total)
```
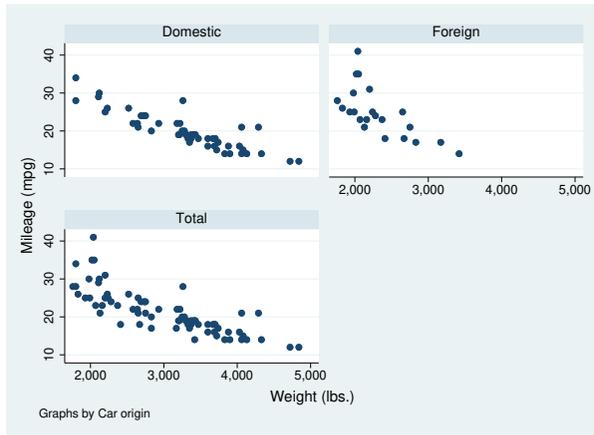

The graphs show that the relationship is nonlinear for both origins of cars.

Syntax note: To make the graphs for the combined subgroups, we ended up using a `by()` option, not a `by` prefix. If we had used a `by` prefix, separate graphs would have been generated instead of the combined graph created by the `by()` option.

Model fitting: Linear regression

After looking at the graphs, we would like to fit a regression model that predicts MPG from the weight and type of the car. From the graphs, we see that the relationship is nonlinear, so we will try modeling MPG as a quadratic in weight. Also from the graphs, we judge the relationship to be different for domestic and foreign cars. We will include an indicator (dummy) variable for `foreign` and evaluate afterward whether this adequately describes the difference. Thus we will fit the model

$$\mathtt{mpg} = \beta_0 + \beta_1\,\mathtt{weight} + \beta_2\,\mathtt{weight}^2 + \beta_3\,\mathtt{foreign} + \epsilon$$

`foreign` is already an indicator (0/1) variable, but we need to create the weight-squared variable. This can be done with the menus, but here using the command line is simpler. Type

```
. generate wtsq = weight^2
```

Now that we have all the variables we need, we can run a linear regression. We will use the menus and see that the command is also simple. To use the menus, select **Statistics > Linear models and related > Linear regression**. In the resulting dialog, choose `mpg` as the *Dependent variable* and `weight`, `wtsq`, and `foreign` as the *Independent variables*. **Submit** the command. Here is the equivalent simple `regress` command and the resulting analysis-of-variance table.

```
. regress mpg weight wtsq foreign

      Source |       SS           df       MS      Number of obs   =        74
-------------+----------------------------------   F(3, 70)        =     52.25
       Model |  1689.15372         3   563.05124   Prob > F        =    0.0000
    Residual |   754.30574        70  10.7757963   R-squared       =    0.6913
-------------+----------------------------------   Adj R-squared   =    0.6781
       Total |  2443.45946        73  33.4720474   Root MSE        =    3.2827

------------------------------------------------------------------------------
         mpg | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
      weight |   -.0165729   .0039692    -4.18   0.000    -.0244892   -.0086567
        wtsq |    1.59e-06   6.25e-07     2.55   0.013     3.45e-07    2.84e-06
     foreign |     -2.2035   1.059246    -2.08   0.041      -4.3161   -.0909002
       _cons |    56.53884   6.197383     9.12   0.000     44.17855    68.89913
------------------------------------------------------------------------------
```

The results look encouraging, so we will plot the predicted values on top of the scatterplots for each of the origins of cars. To do this, we need the predicted, or fitted, values. This can be done with the menus, but doing it in the Command window is simple enough. We will create a new variable, `mpghat`, to hold the predicted MPG for each car. Type

```
. predict mpghat
(option xb assumed; fitted values)
```

The output from this command is simply a notification. Go over to the Variables window and scroll to the bottom to confirm that there is now an `mpghat` variable. If you were to try this command when `mpghat` already existed, Stata would refuse to overwrite your data:

```
. predict mpghat
variable mpghat already defined
r(110);
```

The `predict` command, when used after a regression, is called a *postestimation command.* As specified, it creates a new variable called `mpghat` equal to

$$-0.0165729 \, \texttt{weight} + 1.59 \times 10^{-6} \, \texttt{wtsq} - 2.2035 \, \texttt{foreign} + 56.53884$$

For careful model fitting, there are several features available to you after estimation—one is calculating predicted values. Be sure to read [U] **20 Estimation and postestimation commands**.

We can now graph the data and the predicted curve to evaluate separately the fit on the foreign and domestic data to determine if our shift parameter is adequate. We can draw both graphs together. Using the menus and a dialog, do the following:

1. Select **Graphics > Twoway graph (scatter, line, etc.)** from the menus.

2. If there are any plots listed, click on the **Reset** button, [icon], to clear the dialog box.

3. Create the graph for `mpg` versus `weight`:
    a. Click on the **Create...** button.
    b. Be sure that *Basic plots* and *Scatter* are selected.
    c. Select `mpg` as the *Y variable* and `weight` as the *X variable*.
    d. Click on **Accept**.
4. Create the graph showing `mpghat` versus `weight`:
    a. Click on the **Create...** button.
    b. Select *Basic plots* and *Line*.
    c. Select `mpghat` as the *Y variable* and `weight` as the *X variable*.
    d. Check the *Sort on x variable* box. Doing so ensures that the lines connect from smallest to largest `weight` values, instead of the order in which the data happen to be.
    e. Click on **Accept**.
5. Show two plots, one each for domestic and foreign cars, on the same graph:
    a. Click on the **By** tab.
    b. Check the *Draw subgraphs for unique values of variables* checkbox.
    c. Enter `foreign` in the *Variables* field.
6. Click on the **Submit** button.

Here are the resulting command and graph:

```
. twoway (scatter mpg weight) (line mpghat weight, sort), by(foreign)
```


Graphs by Car origin
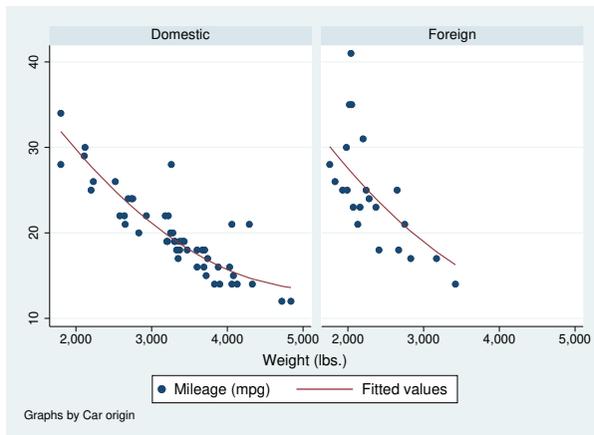
Here we can see the reason for enclosing the separate `scatter` and `line` commands in parentheses: they can thereby be overlaid by submitting them together. The fit of the plots looks good and is cause for initial excitement. So much excitement, in fact, that we decide to print the graph and show it to an engineering friend. We print the graph, being careful to print the graph (and not all our results), by choosing **File > Print...** from the Graph window menu bar.

When we show our graph to our engineering friend, she is concerned. "No," she says. "It should take twice as much energy to move 2,000 pounds 1 mile compared with moving 1,000 pounds the same distance: therefore, it should consume twice as much gasoline. Miles per gallon is not quadratic in weight; gallons per mile is a linear function of weight. Don't you remember any physics?"

We try out what she says. We need to generate an energy-per-distance variable and make a scatterplot. Here are the commands that we would need—note their similarity to commands issued earlier in the session. There is one new command, the `label variable` command, which allows us to give the `gpm100m` variable a *variable label* so that the graph is labeled nicely.

```
. generate gp100m = 100/mpg
. label variable gp100m "Gallons per 100 miles"
. twoway (scatter gp100m weight), by(foreign, total)
```


Graphs by Car origin
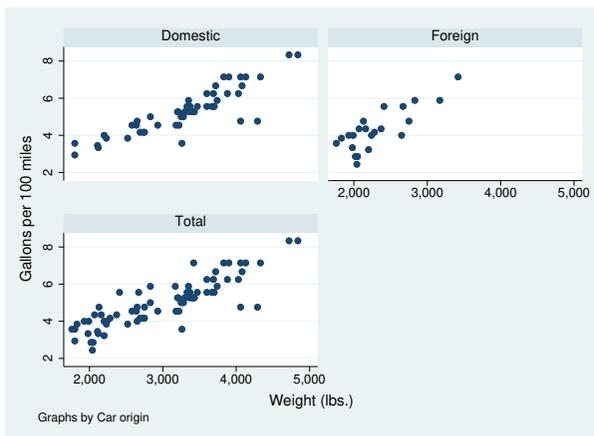
Sadly satisfied that the engineer is indeed correct, we rerun the regression:

```
. regress gp100m weight foreign
      Source |       SS           df       MS      Number of obs   =        74
-------------+----------------------------------   F(2, 71)        =    113.97
       Model |  91.1761694         2  45.5880847   Prob > F        =    0.0000
    Residual |  28.4000913        71  .400001287   R-squared       =    0.7625
-------------+----------------------------------   Adj R-squared   =    0.7558
       Total |  119.576261        73  1.63803097   Root MSE        =    .63246

      gp100m | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
      weight |   .0016254   .0001183    13.74   0.000     .0013896    .0018612
     foreign |   .6220535   .1997381     3.11   0.003     .2237871    1.02032
       _cons |  -.0734839   .4019932    -0.18   0.855    -.8750354    .7280677
```

We find that foreign cars had better gas mileage than domestic cars in 1978 because they were so light. According to our model, a foreign car with the same weight as a domestic car would use an additional 5/8 gallon (or 5 pints) of gasoline per 100 miles driven. With this conclusion, we are satisfied with our analysis.

Commands versus menus

In this chapter, you have seen that Stata can operate either with menu choices and dialogs or with the Command window. As you become more familiar with Stata, you will find that the Command window is typically much faster for oft-used commands, whereas the menus and dialogs are faster when building up complex commands, such as those that create graphs.

One of Stata's great strengths is the consistency of its *command syntax*. Most of Stata's commands share the following syntax, where square brackets mean that something is optional, and a *varlist* is a list of variables.

$$\big[\textit{prefix}\mathbf{:}\big]\ \textit{command}\ \big[\textit{varlist}\big]\ \big[\textit{if}\big]\ \big[\textit{in}\big]\ \big[\textit{weight}\big]\ \big[\mathbf{,}\ \textit{options}\big]$$

Some general rules:

- Most commands accept prefix commands that modify their behavior; see [U] **11.1.10 Prefix commands** for details. One of the common prefix commands is `by`.
- If an optional *varlist* is not specified, all the variables are used.
- *if* and *in* restrict the observations on which the command is run.
- *options* modify what the command does.
- Each command's syntax is found in the system help and the reference manuals.
- Stata's command syntax includes more than we have shown you here, but this introduction should get you started. For more information, see [U] **11 Language syntax** and `help language`.

We saw examples using all the pieces of this except for the `in` qualifier and the `weight` clause. The syntax for all commands can be found in the system help along with examples—see [GSW] **4 Getting help** for more information. The consistent syntax makes it straightforward to learn new commands and to read others' commands when examining an analysis.

Here is an example of reading the syntax diagram that uses the `summarize` command from earlier in this chapter. The syntax diagram for `summarize` is typical:

$$\mathtt{summarize}\ \big[\textit{varlist}\big]\ \big[\textit{if}\big]\ \big[\textit{in}\big]\ \big[\textit{weight}\big]\ \big[\mathbf{,}\ \textit{options}\big]$$

This means that

command by itself is valid:	`summarize`
command followed by a *varlist* (variable list) is valid:	`summarize mpg`
	`summarize mpg weight`
command with *if* (with or without a *varlist*) is valid:	`summarize if mpg>20`
	`summarize mpg weight if mpg>20`

and so on.

You can learn about `summarize` in [R] **summarize**, or select **Help > Stata command...** and enter `summarize`, or type `help summarize` in the Command window.

Keeping track of your work

It would have been useful if we had made a log of what we did so that we could conveniently look back at interesting results or track any changes that were made. You will learn to do this in [GSW] **16 Saving and printing results by using logs**. Your logs will contain commands and their output—another reason to learn command syntax is so that you can remember what you have done.

To make a log file that keeps track of everything appearing in the Results window, click on the **Log** button, which looks like a lab notebook, 📓 . Choose a place to store your log file, and give it a name, just as you would for any other document. The log file will save everything that appears in the Results window from the time you start a log file to the time you close it.

Video example

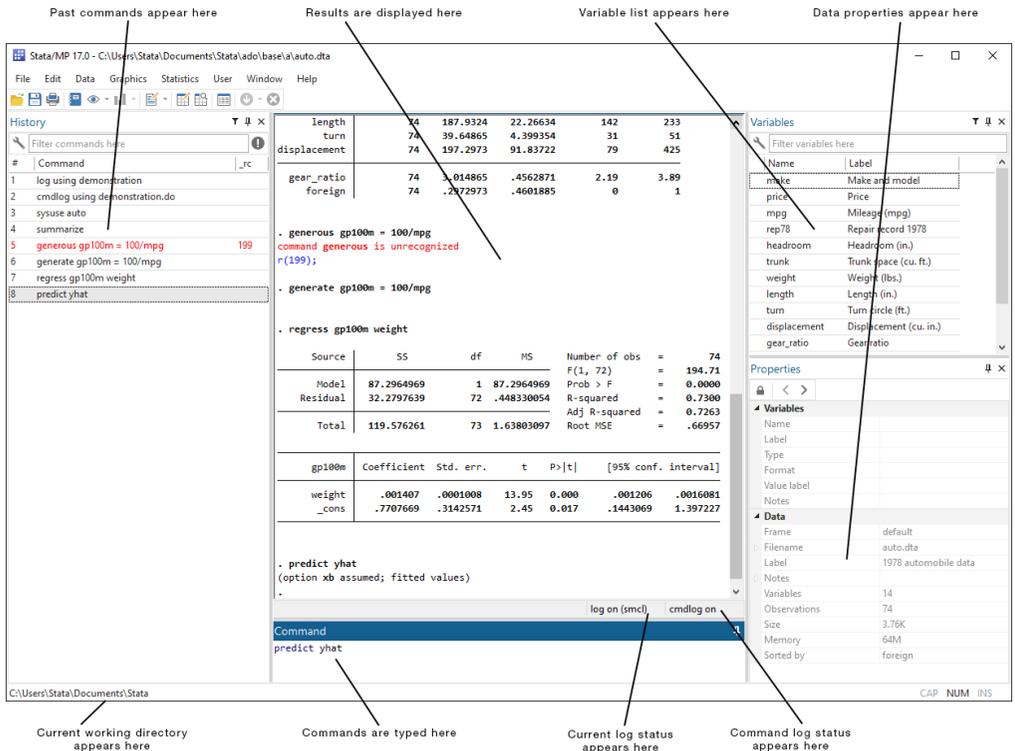What's it like—Getting started in Stata

Conclusion

This chapter introduced you to Stata's capabilities. You should now read and work through the rest of this manual. Once you are done here, you can read the *User's Guide*.

2 The Stata user interface

The windows

This chapter introduces the core of Stata's interface: its main windows, its toolbar, its menus, and its dialogs.

The five main windows are the History, Results, Command, Variables, and Properties windows. Except for the Results window, each window has its name in its title bar. These five windows are typically in use the whole time Stata is open. There are other, more specialized windows such as the Viewer, Data Editor, Variables Manager, Do-file Editor, Graph, and Graph Editor windows—these are discussed later in this manual.

To open any window or to reveal a window hidden by other windows, select the window from the **Window** menu, or select the proper item from the toolbar. You can also use *Ctrl+Tab* to cycle through all open windows inside the main Stata window or *Alt+Tab* to cycle through all open windows (Stata and other) if you want to change windows from the keyboard. Many of Stata's windows have functionality that can be accessed by clicking on the right mouse button (right-clicking) within the window. Right-clicking displays a contextual menu that, depending on the window, allows you to copy text, set the preferences for the window, or print the contents of the window. When you copy text or print, we recommend that you always right-click on the window rather than use the menu bar or toolbar so that you can be sure of where and what you are copying or printing.

The toolbar

This is the toolbar:

The toolbar contains buttons that provide quick access to Stata's more commonly used features. If you forget what a button does, hold the mouse pointer over the button for a moment, and a tooltip will appear with a description of that button.

Buttons that include both an icon and an arrow display a menu if you click on the arrow. Here is an overview of the toolbar buttons and their functions:

Open opens a Stata dataset. Click on the button to open a dataset with the *Open* dialog.

Save saves the Stata dataset currently in memory to disk.

Print displays a list of windows. Select a window name to print its contents.

Log begins a new log or closes, suspends, or resumes the current log. See [GSW] **16 Saving and printing results by using logs** for an explanation of log files.

Viewer opens the Viewer or brings a Viewer to the front of all other windows. Click on the button to open a new Viewer. Click on the arrow to select a Viewer to bring to the front. See [GSW] **3 Using the Viewer** for more information.

Graph brings a Graph window to the front of all other windows. Click on the button to bring the Graph window to the front. Click on the arrow to select a Graph window to bring to the front. See *The Graph button* in [GSW] **14 Graphing data** for more information.

Do-file Editor opens the Do-file Editor or brings a Do-file Editor to the front of all other windows. Click on the button to open a new Do-file Editor. Click on the arrow to select a Do-file Editor to bring to the front. See [GSW] **13 Using the Do-file Editor—automating Stata** for more information.

Data Editor (Edit) opens the Data Editor or brings the Data Editor to the front of the other Stata windows. See [GSW] **6 Using the Data Editor** for more information.

Data Editor (Browse) opens the Data Editor in browse mode. See *Browse mode* in [GSW] **6 Using the Data Editor** for more information.

Variables Manager opens the Variables Manager. See [GSW] **7 Using the Variables Manager** for more information.

Show more results tells Stata to continue when it has paused in the middle of long output. Click on the arrow to choose whether to run the command to completion. See [GSW] **B.8 More** for more information.

Break stops the current task in Stata. See [GSW] **10 Listing data and basic command syntax** for more information.

The Command window

Commands are submitted to Stata from the Command window. The Command window supports basic text editing, copying and pasting, a command history, function-key mapping, filename completion, and variable-name completion.

From the Command window, pressing

Page Up	steps backward through the command history.
Page Down	steps forward through the command history.
Tab	autocompletes a partially typed variable name when possible or presents a list of similar names if there could be more than one completion. Further typing will narrow the list. As soon as the name is complete, the full name will be inserted. If the name starts with a double quote, *Tab* attempts to autocomplete a filename in the same manner.

See [U] **10 Keyboard use** for more information about keyboard shortcuts for the Command window.

The command history allows you to recall a previously submitted command, edit it if you wish, and then resubmit it. Commands submitted by Stata's dialogs are also included in the command history, so you can recall and submit a command without having to open the dialog again.

The Results window

The Results window contains all the commands and their textual results you have entered during the Stata session.

You can scroll through the Results window to look at work you have done, it is much simpler to search within the Results window by using the find bar. By default, the find bar is hidden. You can expose it by selecting **Edit > Find...**.

You can clear out the Results window at any time by right-clicking in the Results window and selecting **Clear results** from the contextual menu. This action is not undoable.

The Variables window

The Variables window shows the list of variables in the dataset, along with the properties of the variables. By default, it shows all the variables and their variable labels. You can change what properties get displayed by right-clicking on the header of any column of the Variables window.

Click once on a variable in the Variables window to select it. Multiple variables can be selected in the usual fashion, either by *Ctrl*-clicking on nonadjacent variables or by clicking on a variable and *Shift*-clicking on a second variable to select all intervening variables.

Double-clicking on a variable in the Variables window puts the selected variable at the insertion point in the Command window.

The leftmost column of the Variables window is called the one-click paste column. You can also send variables to the Command window by hovering the mouse over the one-click paste column of the Variables window and clicking on the arrow that appears. The one-click paste column can be shown or hidden in the same fashion as the other columns in the Variables window.

The Variables window supports filtering and changing the display order of the variables. Text entered in the *Filter variables here* field will filter the variables appearing in the Variables window. The filter is applied to all visible columns and shows all variables that match the criteria in at least one column. By default, the filter will ignore case and show any variables for which at least one column contains any of the words in the filter. Clicking on the wrench on the left will allow you to change this behavior as well as add or remove additional columns containing information about the variables.

You can change the display order of the variables in the Variables window by clicking on any column header. The first click sorts in ascending order, the second click sorts in descending order, and the third click puts the variables back in dataset order. Thus clicking on the *Name* column header will make the Variables window display the variables in alphabetical order. Sorting in the Variables window is live, so if you change a property of a variable when the Variables window is sorted by that property, it will automatically move the variable to its proper location. Reordering the display order of the variables in the Variables window does not affect the order of the variables in the dataset itself.

Right-clicking on a variable in the Variables window displays a menu from which you can select

- **Keep only selected variables** to keep just the selected variables in the dataset in memory. You will be asked for confirmation. This affects only the dataset in memory, not the dataset as saved on your disk. See [GSW] **12 Deleting variables and observations** for more information.
- **Drop selected variables** to drop, or eliminate, the selected variables from the dataset in memory. You will be asked for confirmation. Just as above, this affects only the dataset in memory, not the dataset as saved on your disk. See [GSW] **12 Deleting variables and observations** for more information.
- **Copy varlist** to copy the selected variable names to the clipboard.
- **Select all** to select all variables in the dataset that satisfy the filter conditions. If no filter has been specified, all variables will be selected.
- **Send varlist to Command window** to send all selected variables to the Command window.
- **Font...** to bring up a *Font* dialog, allowing you to change the font used to display the Variables window contents.

Items from the contextual menu issue standard Stata commands, so working by right-clicking is just like working directly in the Command window.

If you would like to hide the Variables window, click on its close button.

To reveal a hidden Variables window, select **Window > Variables**.

The Properties window

The Properties window displays variable and dataset properties. If a single variable is selected in the Variables window, its properties are displayed. If there are multiple variables selected in the Variables window, the Properties window will display properties that are common across all selected variables.

Clicking the lock icon in the Properties window title bar toggles the ability to alter properties of the selected variables. By default, changes are not allowed. Once the properties are unlocked, you can make any changes to variable or dataset properties you like. Each change you make will create a command that appears in the Results and Command windows, as well as in any command log, so the changes are reproducible. Using the Properties window is one of the simplest ways of managing notes, changing variable and value labels, and changing display formats. See [D] **notes**, [D] **label**, and [D] **format**.

Clicking the arrow buttons next to the lock icon will select the previous or next variable shown in the Variables window, and that selection will be reflected in the Properties window. If you would like to hide the Properties window, click on its close box. If you would like to reveal a hidden Properties window, select **Window > Properties**.

You should also investigate the Variables Manager, explained in [GSW] **7 Using the Variables Manager**, because it extends these capabilities and provides a good interface for managing variables.

The History window

The History window shows the history of commands that have been entered, with unsuccessful commands and their error codes in red, by default.

The toolbar has two tools for manipulating the contents of the History window. Clicking on the

Filter button, ▼ , in the History window title bar toggles the visibility of these tools. Text entered in the *Filter commands here* field will filter the commands appearing in the History window. By default, the filter ignores case and finds any commands containing any of the words in the filter. Clicking on the wrench on the left allows you to change this behavior. You can hide the commands that produced

an error by clicking on the **Filter errors** button, ❗ .

To enter a command from the History window, you can
- Click once on a past command to copy it to the Command window, replacing the contents of the Command window.
- Double-click on a past command to resubmit it. Executing the command adds the command to the bottom of the History window.

Right-clicking on the History window displays a menu from which you can select various actions:
- **Cut** removes the selected commands from the History window and places them on the Clipboard.
- **Copy** copies the selected commands to the Clipboard.
- **Delete** removes the selected commands from the History window.
- **Select all** selects all the commands in the History window, including those before and after the commands currently displayed.
- **Clear all** clears out all the commands from the History window, including those before and after the commands currently displayed.
- **Do selected** submits all the selected commands and adds them to the bottom of the command history. Stata will attempt to run all the selected commands, even those containing errors, and will not stop even if a command causes an error.
- **Send selected to Do-file Editor** places all the selected commands into a new Do-file Editor window.
- **Save all...** brings up a *Save review contents* dialog, which allows you to save all the commands in the History window, including those before and after the commands currently displayed, in a *do-file.* (See [GSW] **13 Using the Do-file Editor—automating Stata** for more information on do-files.)
- **Save selected...** brings up a *Save review contents* dialog, which allows you to save the selected commands in the History window in a do-file.
- **Font...** brings up a *Font* dialog, allowing you to change the font used to display the History window contents.

Menus and dialogs

There are two ways by which you can tell Stata what you would like it to do: you can use menus and dialogs, or you can use the Command window. When you worked through the sample session in [GSW] **1 Introducing Stata—sample session**, you saw that both ways have strengths. We will discuss the menus and dialogs here.

Stata's **Data**, **Graphics**, and **Statistics** menus provide point-and-click access to almost every command in Stata. As you will learn, Stata is fully programmable, and Stata users can even create their own dialogs and menus. The **User** menu provides a place for programmers to add their own menu items. Initially, it contains only some empty submenus. As an example, suppose you wish to perform a Poisson regression. You could type Stata's `poisson` command, or you could select **Statistics > Count outcomes > Poisson regression**, which would display this dialog:
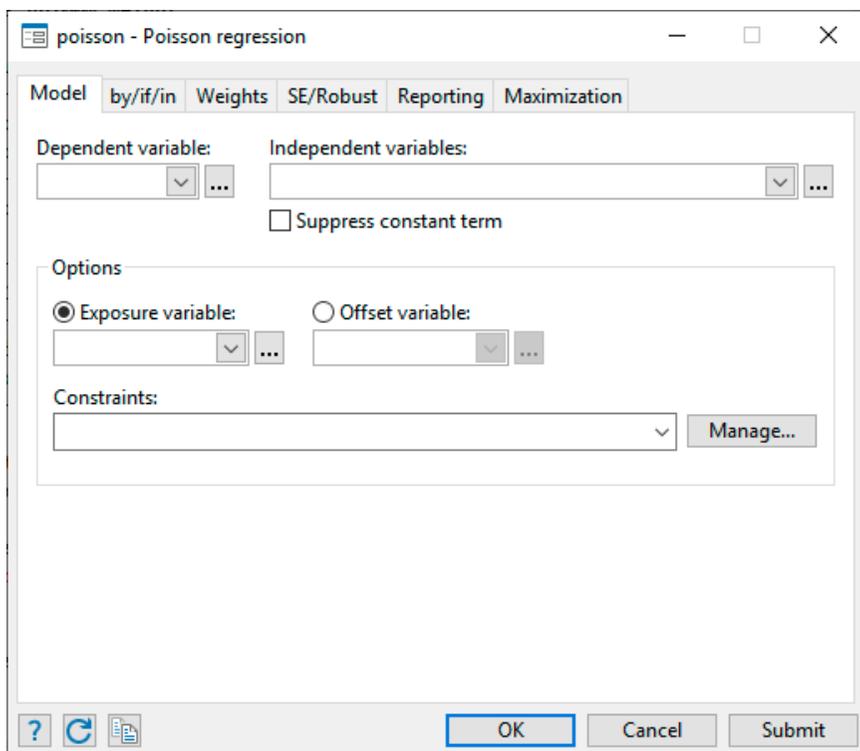
This dialog provides access to all the functionality of Stata's `poisson` command. Because the dependent and independent variables must be numeric, you will find that the combo box will display only numeric variables for choosing. The `poisson` command has many options that can be accessed by clicking on the multiple tabs across the top of the dialog. The first time you use the dialog for a command, it is a good idea to look at the contents of each tab so that you will know all the dialog's capabilities.

The dialogs for many commands have the **by/if/in** and **Weights** tabs. These provide access to Stata's commands and qualifiers for controlling the estimation sample and dealing with weighted data. See [U] **11 Language syntax** for more information on these features of Stata's language.

The dialogs for most estimation commands have the **Maximization** tab for setting the maximization options (see [R] **Maximize**). For example, you can specify the maximum number of iterations for the optimizer.
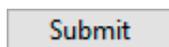
Most dialogs in Stata provide the same six buttons you see at the bottom of the *poisson* dialog above.
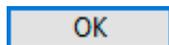
OK issues a Stata command based on how you have filled out the fields in the dialog and then closes the dialog.

Cancel closes the dialog without doing anything—just as clicking on the dialog's close button does.

Submit issues a command just like **OK** but leaves the dialog on the screen so that you can make changes and issue another command. This feature is handy when, for example, you are learning a new command or putting together a complicated graph.

Help provides access to Stata's help system. Clicking on this button will typically take you to the help file for the Stata command associated with the dialog. Clicking on it here would take you to the poisson help file. The help file will have tabs above groups of options to show which dialog tab contains which options.

Reset resets the dialog to its default state. Each time you open a dialog, it will remember how you last filled it out. If you wish to reset its fields to their default values at any time, simply click on this button.

Copy command to Clipboard behaves much like the **Submit** button, but rather than issuing a command, it copies the command to the Clipboard. The command can then be pasted elsewhere (such as in the Do-file Editor).

The command issued by a dialog is submitted just as if you had typed it by hand. You can see the command in the Results window and in the History window after it executes. Looking carefully at the full command will help you learn Stata's command syntax.

In addition to being able to access the dialogs for Stata commands through Stata's menus, you can also invoke them by using two other methods. You may know the name of a Stata command for which you want to see a dialog, but you might not remember how to navigate to that command in the menu system. Simply type db *commandname* to launch the dialog for *commandname*:

```
. db poisson
```

You will also find access to the dialog for a command in that command's help file; see [GSW] **4 Getting help** for more details.

As you read this manual, we will present examples of Stata commands. You may type those examples as presented, but you should also experiment with submitting those commands by using their dialogs. Use the db command described above to quickly launch the dialog for any command that you see in this manual.

The working directory

If you look at the screenshot on page 23, you will notice the status bar at the base of the main Stata window that contains the name of the current working directory you set when you installed Stata. We will use the directory C:\Users\Stata\Documents\Stata in our examples; your exact directory will differ. The current working directory is the folder where graphs and datasets will be saved when typing commands such as save *filename*. It does not affect the behavior of menu-driven file actions such as **File > Save** or **File > Open...**. Once you have started Stata, you can change the current working directory with the cd command. See [D] **cd** for full details. Stata always displays the name of the current working directory so that it is easy to tell where your graphs and datasets will be saved.

Fine control of Stata's windows

When Stata is first launched, the History, Results, Command, Variables, and Properties windows appear within the main Stata window. The History, Variables, Properties, and Command windows are initially linked to one another and attached to the edges of the Stata window. They can be resized by dragging on their edges. They even have two more features: you can drag them by their title bars to reposition them within the main window, and you can hide them as tabs when not in use. To hide them as tabs, click on the pushpin icons in their title bars.

The Results window always occupies whatever space remains inside the main Stata window.

All other windows that open up can be moved freely and independently of the Stata window.

This default behavior is simple and familiar and needs little explanation. Try playing with the windows inside the main Stata window now. If you find that the window behavior is good, you can stop reading this section. If you find that you would like some extra customization or that you would simply like some technical explanation of added window behaviors, keep reading.

Window types

The Stata for Windows interface has two types of windows: *docking* and *nondocking*. The History, Command, Variables, and Properties windows are docking windows. All other windows are nondocking.

Docking windows have special characteristics that allow them to be used with other docking windows: they can be *linked* (sharing a window with a splitter dividing the windows) or *tabbed* (sharing a window with a tab for each docking window). They can be set to automatically hide when not needed. Nondocking windows have none of these special characteristics.

Here is a brief comparison of the two types of windows:

Docking windows
- can be linked to other docking windows so that they share a window with a splitter dividing them;
- can share one window with other docking windows with a tab for each; and
- can be made to hide automatically when not in use.

Nondocking windows
- are separate from the main Stata window (except for the Results window);
- cannot be linked to other windows;
- cannot be tabbed to another window; and
- cannot be made to hide automatically when not in use.

When Stata is first launched, the History, Results, Command, Variables, and Properties windows appear within the main Stata window. The History, Command, Variables, and Properties windows are docking windows and are initially both docked and linked to one another. The Results window is a nondocking window.

Docking windows

Docking windows may be moved within the main Stata window by dragging on their title bars. When you drag a docking window over the main Stata window or another docking window, docking guides appear (see figure 1). Dragging the title bar over a docking guide gives you a preview of how and where the window will appear if you release the mouse button.

Dropping the box on one of the outer docking guides links the docking window to the window under the guide. For example, figure 1 shows how you would drag the History window over the Variables window, at which point the docking guides appear. When you move the mouse over the bottom docking guide and release the mouse button, the History window becomes linked to the Variables window and is displayed in the lower part of the window.

When docking windows are linked, a divider (splitter) is inserted between them. The splitter allows the linked windows to be resized; increasing the size of one window decreases the size of the other.
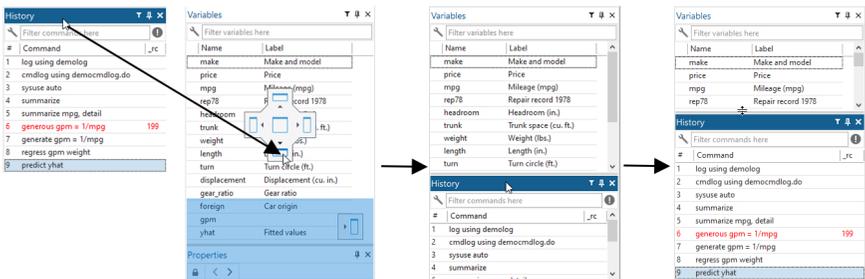

Figure 1: Linking two windows

Dropping the box on the center docking guide (see figure 2) combines the docking window with the window under the guide. One docking window is created with a tab for each docked window at the bottom of the new window. Clicking on a tab displays its window. To undock a tabbed window, click and drag the window's tab out of the docking window to a docking guide. The docking window containing the tabbed windows can also be linked to other docking windows.
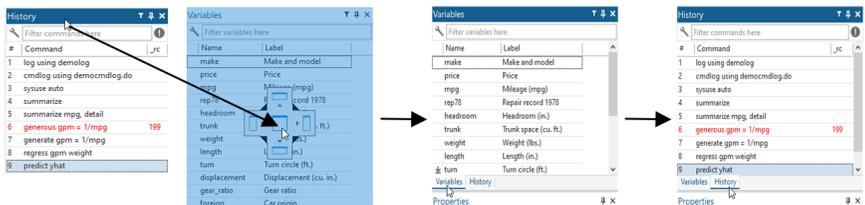

Figure 2: Docking two windows

Auto Hide and pinning

Docking windows support the Auto Hide feature, which automatically hides the docking windows when they are not in use. Pushpins appear in the title bar of all docked windows in the main Stata window.

To enable Auto Hide for a docking window, click on the pushpin in the title bar (see figure 3) so that the pushpin is horizontal. Click on the pushpin again to disable Auto Hide; the pushpin will be vertical.

After you enable Auto Hide for a docking window, the window is hidden when not in use and is displayed as a tab along an edge of the main Stata window. To show the window, move the mouse pointer over the tab. To hide the window again, move the mouse pointer outside the window. If the window has the focus (its title bar is highlighted), it will be hidden if you click on another window.
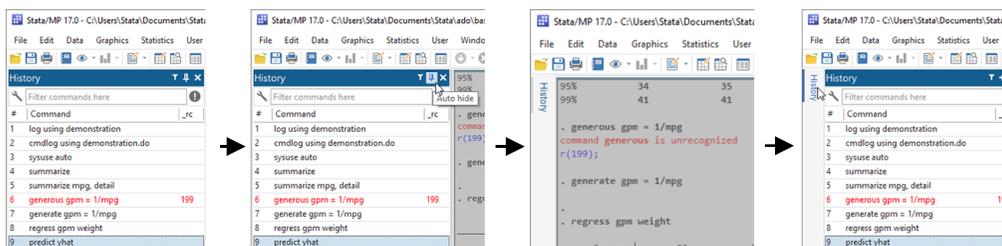

Figure 3: Enabling Auto Hide

If you enable Auto Hide for the History window in its default docked location, it will obscure any text at the beginning of the Command window when it is displayed. To avoid this problem, dock this window at the right edge of the main Stata window before enabling Auto Hide.

Nondocking windows

Nondocking windows include the Graph window, Data Editor, Do-file Editor, Viewers, and dialogs. These windows are always independent of the main Stata window. The Results window, because of its status as the primary window of Stata, must remain inside the main Stata window.

3 Using the Viewer

The Viewer's purpose

The Viewer is a versatile tool in Stata. It will be the first place you can turn for help within Stata, but it is far more than just a help system. You can also use the Viewer to add, delete, and manage third-party extensions to Stata that are known as *community-contributed features*; to view and print Stata logs from both your current and your previous Stata sessions; to view and print any other Stata-formatted (SMCL) or plain-text file; and even to launch your web browser to follow hyperlinks.

This chapter focuses on the general use of the Viewer, its buttons, and a brief summary of the commands that the Viewer understands. There is more information about using the Viewer to find help in [GSW] **4 Getting help** and for installing community-contributed features in [GSW] **19 Updating and extending Stata—Internet functionality**.

To open a new Viewer window (or open a new tab in an existing Viewer), you can either click on the **Viewer** button, 👁 , or select **Window > Viewer > New Viewer**.

Viewer buttons

The toolbar of the Viewer has multiple buttons, a command box, and a search box.

←	**Back** goes back one step in your viewing trail.
→	**Forward** goes forward one step in your viewing trail, assuming you backtracked.
↻	**Reload page** refreshes the Viewer, in case you are viewing something that has changed since you opened the Viewer.
🖶	**Print** prints the contents of the Viewer.
🔍	**Find text in page** opens the find bar at the bottom of the Viewer (see below).
🔍⏷ [ ]	**Search** chooses the scope of help searches in the Viewer.

The Find bar is used to find text within the current Viewer. To reveal the Find bar at the bottom of the window, click on the **Find text in page** button:
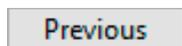
The Find bar has its own buttons, fields, and checkboxes.

Close closes the Find bar.

Find is the field for entering the search text you would like to find. You can change the search options by using the checkboxes.

Previous jumps to the previous instance of the search text; it automatically wraps past the start of the Viewer document if there are no previous instances of the search text.
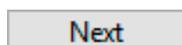
Next jumps to the next instance of the search text; it automatically wraps past the end of the Viewer document if there are no further instances of the search text.

Highlight all highlights other instances of the search text (in yellow, by default) when this box is checked. If unchecked, only the current instance of the search text is highlighted (in black, by default). By default, this box is checked.

Match case, when checked, considers uppercase and lowercase letters to be different. When this box is checked, searching for This would not find this. If unchecked, uppercase and lowercase letters are considered the same, so searching for This would find this. By default, this box is unchecked.

Viewer's function

The Viewer is similar to a web browser. It has links (shown in blue text by default) that you can click on to see related help topics and to install and manage third-party software. When you move the mouse pointer over a link, the status bar at the bottom of the Viewer shows the action associated with that link. If the action of a link is `help logistic`, clicking on that link will show the help file for the `logistic` command in the Viewer. Middle-clicking on a link in a Viewer window (*Ctrl*+clicking if you do not have a three-button mouse) will open the link in a new tab in the Viewer window. *Shift*+clicking will open the link in a new Viewer window.

You can open a new Viewer by selecting **Window > Viewer > New Viewer** or by clicking on the **Viewer** button on the toolbar of the main window. Entering a `help` command from the Command window will also open a new Viewer.

To bring a Viewer to the front of all other Viewers, select **Window > Viewer** and choose a Viewer from the list there. Selecting **Close all Viewers** closes all open Viewer window and tabs.

Viewing local text files, including SMCL files

In addition to viewing built-in Stata help files, you can use the Viewer to view Stata Markup and Control Language (SMCL) files such as those typically produced when logging your work (see [GSW] **16 Saving and printing results by using logs**) as well as plain-text files. To open a file and view its contents, simply select **File > Open...**, and you will be presented with a dialog:
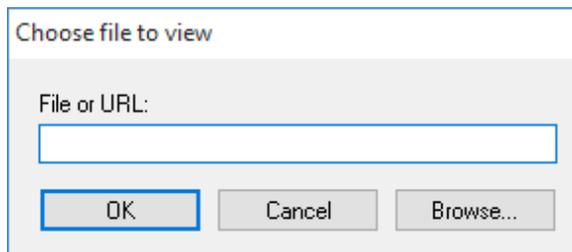
You may either type in the name of the file that you wish to view and click on **OK**, or you may click on the **Browse...** button to open a standard file dialog that allows you to navigate to the file.

If you currently have a log file open, you may view the log file in the Viewer. This method has one advantage over scrolling back in the Results window: what you view stays fixed even as output is added to the Results window. If you wish to view a current log file, select **File > Log > View...**, and the usual dialog will appear but with the path and filename of the current log already in the field. Simply click on **OK**, and the log will appear in the Viewer. See [GSW] **16 Saving and printing results by using logs** for more details.

Viewing remote files over the Internet

If you want to look at a remote file over the Internet, the process is similar to viewing a local file, only instead of using the **Browse...** button, you type the URL of the file that you want to see, such as https://www.stata.com/man/readme.smcl. You should use the Viewer only to view text or SMCL files. If you enter the URL of, say, an arbitrary webpage, you will see the HTML source of the page instead of the usual browser rendering.

Navigating within the Viewer

In addition to using the scrollbar to navigate the Viewer window, you also can use the up and down arrow keys and *Page Up* and *Page Down* keys to do the same. Pressing the up or down arrow key scrolls the window a line at a time. Pressing the *Page Up* or *Page Down* key scrolls the window a screen at a time.

Printing

To print the contents of the Viewer, right-click on the window and select **Print...**. You may also select **File > Print >** *Viewer name* or click on the **Print** toolbar button, 🖶 , to print.

Tabs in the Viewer

A Viewer window can have multiple tabs. You may view different files or different views of the same file in different tabs. Clicking on the **Open new tab** button, ✛ , will open a new tab in the current Viewer window. You can change the order of the tabs within a Viewer by dragging the tabs along the tab bar within the window. If you drag a tab and drop it within the body of the same Viewer window, a menu will appear. If you select **New Horizontal Tab Group**, the Viewer window will be split horizontally. This is useful for having two views of the same file at different places in the file. Similarly, if you select **New Vertical Tab Group**, the Viewer window will be split vertically. This is useful for comparing two similar files side by side.

Once you have created a horizontal or vertical tab group, if you drop a tab inside a group, you will get the choice of creating a similar new group or moving the tab to the selected group.

Right-clicking on the Viewer window

Right-clicking on the Viewer window displays a contextual menu that offers these options:
- **Select all** to select all text in the Viewer.
- **Preferences...** to edit the preferences for the Viewer window.
- **Font...** to change the font for the window.
- **Print...** to print the contents of the Viewer window.

In other contexts, there could be more items displayed in the contextual menu.

Searching for help in the Viewer

The search box in the Viewer can be used to search documentation. Click on the magnifying glass; choose *Search all*, *Search documentation and FAQs*, or *Search net resources*; and then type a word or phrase in the search box and press *Enter*. For more extensive information about using the Viewer for help, see [GSW] **4 Getting help**.

Commands in the Viewer

Everything that can be done in the Viewer by clicking on links and buttons can also be done by typing commands in the command box at the top of the window or on the Stata command line. Some tasks that can be performed in the Viewer are

- obtaining help (see [GSW] **4 Getting help**):

    Type contents to view the contents of Stata's help system.

    Type *commandname* to view the help file for a Stata command.

    Type *keyword* to search documentation, FAQs, and net resources for a topic.

- searching (see [GSW] **4 Getting help**):

    Type search *keyword* to search documentation, FAQs, and net resources for a topic.

    Type search *keyword*, local to search only documentation and FAQs for a topic.

    Type search *keyword*, net to search only net resources for a topic.

- finding and installing community-contributed commands (see [GSW] **4 Getting help** and [GSW] **19 Updating and extending Stata—Internet functionality**):

    Type net from https://www.stata.com/ to find and install *Stata Journal*, *Stata Technical Bulletin*, and community-contributed commands from the Internet.

    Type ado to review community-contributed packages you have installed.

    Type ado uninstall to uninstall community-contributed packages you have installed on your computer.

- viewing files in the Viewer:

    Type view *filename*.smcl to view SMCL files.

    Type view *filename*.txt to view text files.

    Type view *filename*.log to view text log files.

- viewing files in the Results window:

    Type type *filename*.smcl in the Command window to view SMCL files in the Results window.

    Type type *filename*.txt in the Command window to view text files in the Results window.

    Type type *filename*.log in the Command window to view text log files in the Results window.

- launching your browser to view an HTML file:

    Type browse *URL* to launch your browser.

Using the Viewer from the Command window

Typing help *commandname* in the Command window will bring up a new Viewer showing the requested help.

4 Getting help

System help

Stata's help system provides a wealth of information to help you learn and use Stata. To find out which Stata command will perform the statistical or data management task you would like to do, you should generally follow these steps:

1. Select **Help > Search...**, choose *Search all*, and enter the topic or keywords. This search will open a new Viewer window containing information about Stata commands, references to articles in the *Stata Journal*, links to Frequently Asked Questions (FAQs) on Stata's website, links to videos on Stata's YouTube channel, links to selected external websites, and links to community-contributed features.

2. Read through the results. If you find a useful command, click on the link to the appropriate command name to open its help file.

3. Read the help file for the command you chose.

4. If you want more in-depth help, click on the link from the name of the command to the PDF documentation, read it, then come back to Stata.

5. If the first help file you went to is not what you wanted, either click on the **Also see** menu and choose a link to related help files or click on the **Back** button to go back to the previous document and go from there to other help files.

6. With the help file open, click on the Command window and enter the command, or click on the **Dialog** button and choose a link to open a dialog for the command.

7. If, at any time, you want to begin again with a new search, enter the new search terms in the search box of the Viewer window.

8. If you select *Search documentation and FAQs*, Stata searches its keyword database for official Stata commands, *Stata Journal* articles and software, FAQs, and videos. If you select *Search net resources*, Stata searches for community-contributed commands, whether they are from the *Stata Journal* or elsewhere; see [GSW] **19 Updating and extending Stata—Internet functionality** for more information.
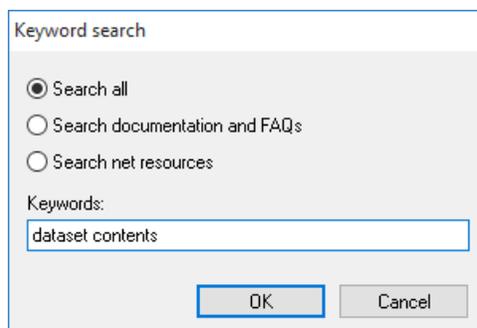
Let's illustrate the help system with an example. You will get the most benefit from the example if you work along at your computer.

Suppose that we have been given a dataset about antique cars and that we need to know what it contains. Though we still have a vague notion of having seen something like this while working through the example session in [GSW] **1 Introducing Stata—sample session**, we do not remember the proper command.
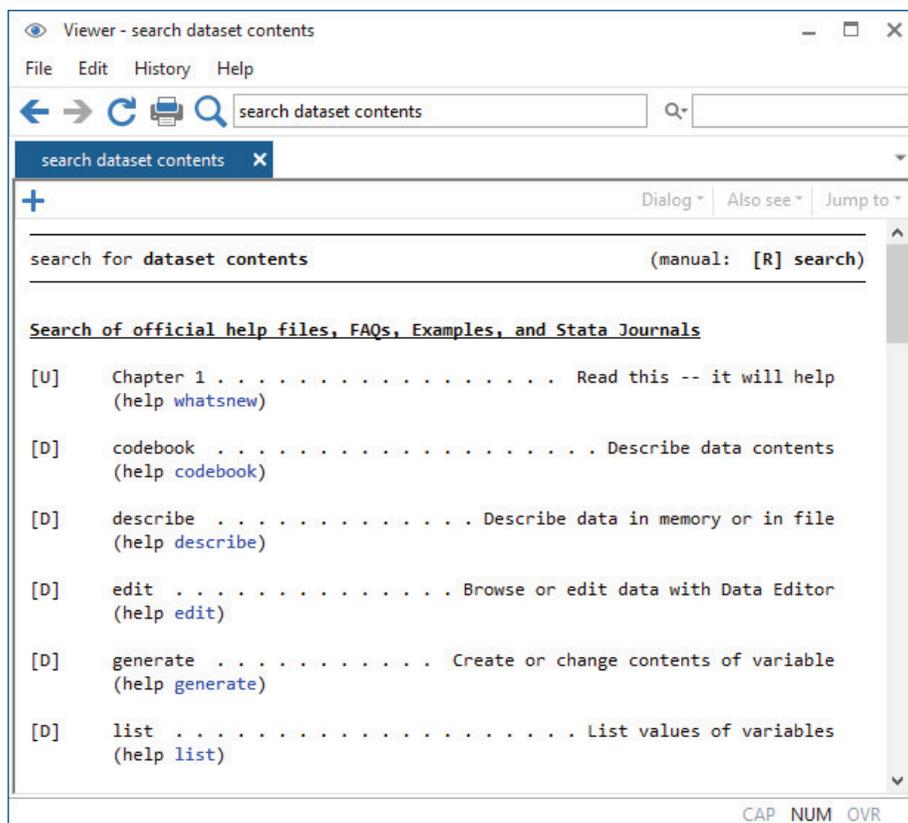
Start by typing `sysuse auto, clear` in the Command window to bring the dataset into memory. (See [GSW] **5 Opening and saving Stata datasets** for information on the `clear` option.)

Follow the above approach:

1. Select **Help > Search...**.

2. Check that the *Search all* radio button is selected.

3. Type `dataset contents` into the search box and click on **OK** or press *Enter*. Before we press *Enter*, the window should look like

Keyword search

⦿ Search all

◯ Search documentation and FAQs

◯ Search net resources

Keywords:

dataset contents

OK   Cancel

4. Stata will now search for "dataset contents" among the Stata commands, the reference manuals, the *Stata Journal*, the FAQs on Stata's website, and community-contributed features. Here is the result:

◉ Viewer - search dataset contents   — ☐ ✕

File   Edit   History   Help

← → C 🖶 Q   search dataset contents   Q▾

search dataset contents   ✕   ▾

\+   Dialog ▾ │ Also see ▾ │ Jump to ▾

search for **dataset contents**   (manual: [R] search)

Search of official help files, FAQs, Examples, and Stata Journals

[U]   Chapter 1 . . . . Read this -- it will help
        (help whatsnew)

[D]   codebook . . . . Describe data contents
        (help codebook)

[D]   describe . . . . Describe data in memory or in file
        (help describe)

[D]   edit . . . . Browse or edit data with Data Editor
        (help edit)

[D]   generate . . . . Create or change contents of variable
        (help generate)

[D]   list . . . . List values of variables
        (help list)

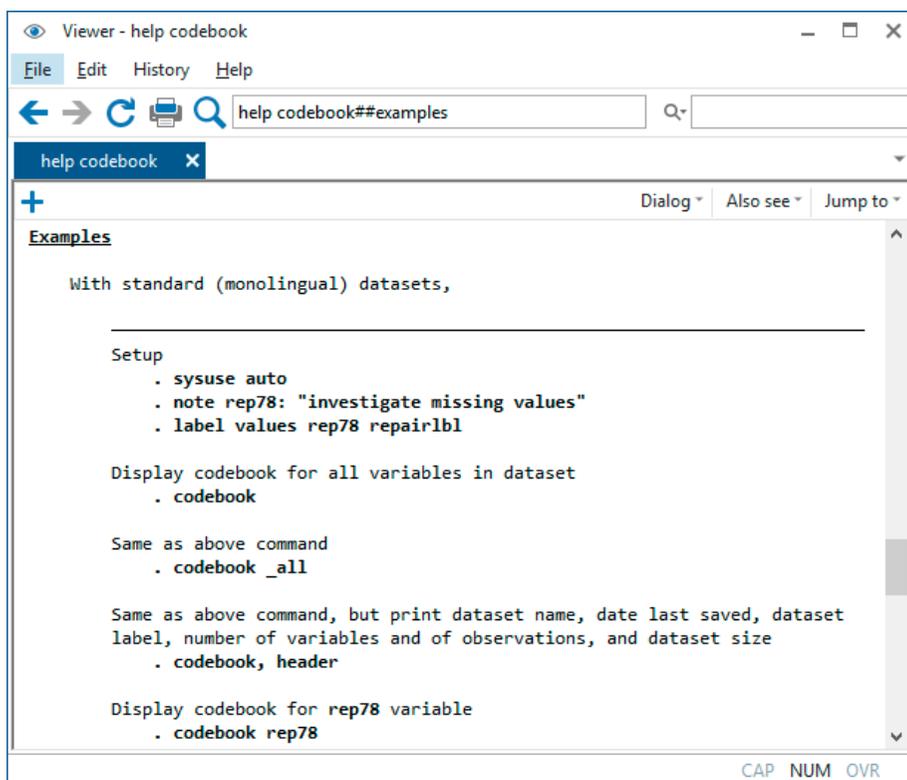   CAP NUM OVR

5. Upon seeing the results of the search, we see two commands that look promising: `codebook` and `describe`. Because we are interested in the contents of the dataset, we decide to check out the `codebook` command. The [D] means that we could look up the `codebook` command in the *Data Management Reference Manual*. The `codebook` link in (`help codebook`) means that there is a system help file for the `codebook` command. This is what we are interested in right now.

6. Click on the `codebook` link. Links can take you to a variety of resources, such as help for Stata commands, dialogs, and even webpages. Here the link goes to the help file for the `codebook` command.
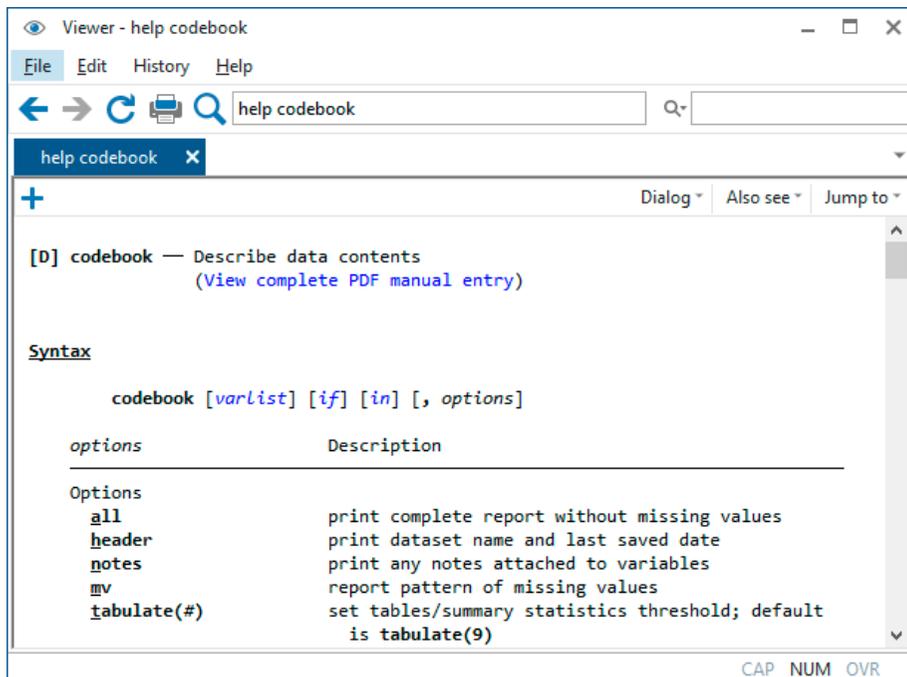
7. What is displayed is typical for help for a Stata command. Help files for Stata commands contain, from top to bottom, these features:

   a. The quick access toolbar with three buttons:
      i. The **Dialog** button shows links to any dialogs associated with the command.
      ii. The **Also see** button shows links to related PDF documentation and help files.
      iii. The **Jump to** button shows links to other sections within the current help file.

   b. The second line of a help file shows a `View complete PDF manual entry` link. Clicking on the link will open the complete documentation for the command—in this case, `codebook`—in your PDF viewer.

   c. The command's syntax, that is, rules for constructing a command that Stata will correctly interpret. The square brackets here indicate that all the arguments to `codebook` are optional but that if we wanted to specify them, we could use a *varlist*, an `if` qualifier, or an `in` qualifier, along with some options. (Options vary greatly from command to command.) The options are listed directly under the command and are explained in some detail later in the help file. You will learn more about command syntax in [GSW] **10 Listing data and basic command syntax**.

d. A description of the command. Because "codebook" is the name for big binders containing a hard copy describing each of the elements of a dataset, the description for the `codebook` command is justifiably terse.

e. The options that can be used with this command. These are explained in much greater detail than in the listing of the possible options after the syntax. Here, for example, we can see that the `mv` option can look to see if there is a pattern in the missing values—something important for data cleaning and imputation.

f. Examples of command usage. The `codebook` examples are real examples that step through using the command on a dataset either shipped with Stata or loadable within Stata from the Internet.

g. The information the command stores in the returned results. These results are used primarily by programmers.

For now, either click on **Jump to** and choose **Examples** from the drop-down menu or scroll down to the examples. It is worth going through the examples as given in the help file. Here is a screenshot of the top of the examples:

```
Viewer - help codebook                                          —  □  ✕
File   Edit   History   Help
←  →  C  🖶  Q  | help codebook##examples          |    Q▾ |              |

| help codebook   ✕ |                                              ▾

+                                          Dialog ▾  Also see ▾  Jump to ▾
Examples                                                              ⌃

    With standard (monolingual) datasets,

        _____

        Setup
            . sysuse auto
            . note rep78: "investigate missing values"
            . label values rep78 repairlbl

        Display codebook for all variables in dataset
            . codebook

        Same as above command
            . codebook _all

        Same as above command, but print dataset name, date last saved, dataset
        label, number of variables and of observations, and dataset size
            . codebook, header

        Display codebook for rep78 variable
            . codebook rep78                                          ⌄

                                                        CAP  NUM  OVR
```

Searching help

Search is designed to help you find information about statistics, graphics, data management, and programming features in Stata, either as part of the official release or as community-contributed features. When entering topics for the search, use appropriate terms from statistics, etc. For example, you could enter `Mann-Whitney`. Multiple topic words are allowed, for example, `regression residuals`.

When you are using **Search**, use proper English and proper statistical terminology. If you already know the name of the Stata command and want to go directly to its help file, select **Help > Stata command...** and type the command name. You can also type the command name in the *Search* field at the top of the Viewer and press *Enter*.

Help distinguishes between topics and Stata commands because some names of Stata commands are also general topic names. For example, logistic is a Stata command. If you choose **Stata command...** and type logistic, you will go right to the help file for the command. But if you choose **Search...** and type logistic, you will get search results listing the many Stata commands that relate to logistic regression.

Remember that you can search for help from within a Viewer window by typing a command in the command box of the Viewer or by clicking the magnifying glass button to the left of the search box, selecting the scope of your search, typing the search criteria in the search box, and pressing *Enter*.

Help and search commands

As you might expect, the help system is accessible from the Command window. This feature is especially convenient when you need help on a particular Stata command. Here is a short listing of the various commands you can use:

- Typing help *commandname* is equivalent to selecting **Help > Stata command...** and typing *commandname*. The help file for the command appears in a new Viewer window.
- Typing search *topic* in the Command window produces the same output as selecting **Help > Search...**, choosing *Search all*, and typing *topic*. The output appears in a new Viewer window.
- Typing search *topic* , local in the Command window produces the same output as selecting **Help > Search...**, choosing *Search documentation and FAQs*, and typing *topic*. The output appears in the Results window instead of a Viewer.
- Typing search *topic* , net in the Command window produces the same output as selecting **Help > Search...**, choosing *Search net resources*, and typing *topic*. The output appears in the Results window instead of a Viewer.

See [U] **4 Stata's help and search facilities** and [U] **4.8 search: All the details** in the *User's Guide* for more information about these command-language versions of the help system. The search command, in particular, has a few capabilities (such as author searches) that we have not demonstrated here.

The Stata reference manuals and User's Guide

All the Stata reference manuals come as PDF files and are included with the software. The manuals themselves have many cross-references in the form of clickable links, so you can easily read the documentation in a nonlinear way.

Many of the links in the help files point to the PDF manuals that came with Stata. It is worth clicking on these links to read the extensive information found in the manuals. The Stata help system, though extensive, contains only a fraction of the information found in the manuals.

Most Stata reference manuals are each arranged alphabetically. Each *Getting Started with Stata* has its own index. A combined index for all other manuals can be found in the *Stata Index*. This combined index is a good place to start when you are looking for information about a command.

Entries have names like **collapse**, **egen**, and **summarize**, which are generally themselves Stata commands.

Notations such as [R] **ci**, [R] **regress**, and [R] **ttest** in the **Search** results and help files are references to the *Base Reference Manual*. You may also see things like [P] **PyStata integration**, which is a reference to the *Programming Reference Manual*, and [U] **12 Data**, which is a reference to the *User's Guide*. For a complete list of manuals and their shorthand notations, see *Cross-referencing the documentation*, which immediately follows the table of contents in this manual.

For advice on how to use the reference manuals, see [GSW] **18 Learning more about Stata**, or see [U] **1.2 The User's Guide and the Reference manuals**.

Stata videos

The Stata YouTube channel is an excellent resource for learning about Stata. The brief videos demonstrate many topics using Stata's graphical user interface. They cover basic topics, such as data management, graphics, summary statistics, and hypothesis testing, and advanced topics, such as multilevel models and structural equation models.

There are also several playlists that provide a series of videos about a topic in sequence. For example, the "Power and sample size calculations" playlist includes videos about how to calculate power, sample size, and effect size for two independent proportions and for paired samples. The "Survival analysis" playlist takes you through the process of setting your data up for survival analysis, conducting basic descriptive analysis of survival data, graphing survival data, and calculating survivor functions and life tables. The "Time series" playlist takes you through the process of setting your data up for time-series analysis, creating time-series graphs, using time-series operators in estimation, and fitting ARMA and ARIMA models. There is even a "Back-to-school video" playlist for students who are using Stata for the first time or want a refresher after summer break.

See https://www.stata.com/links/video-tutorials/ for an up-to-date list of videos organized by topic. The playlists can be accessed directly at https://www.youtube.com/user/statacorp/.
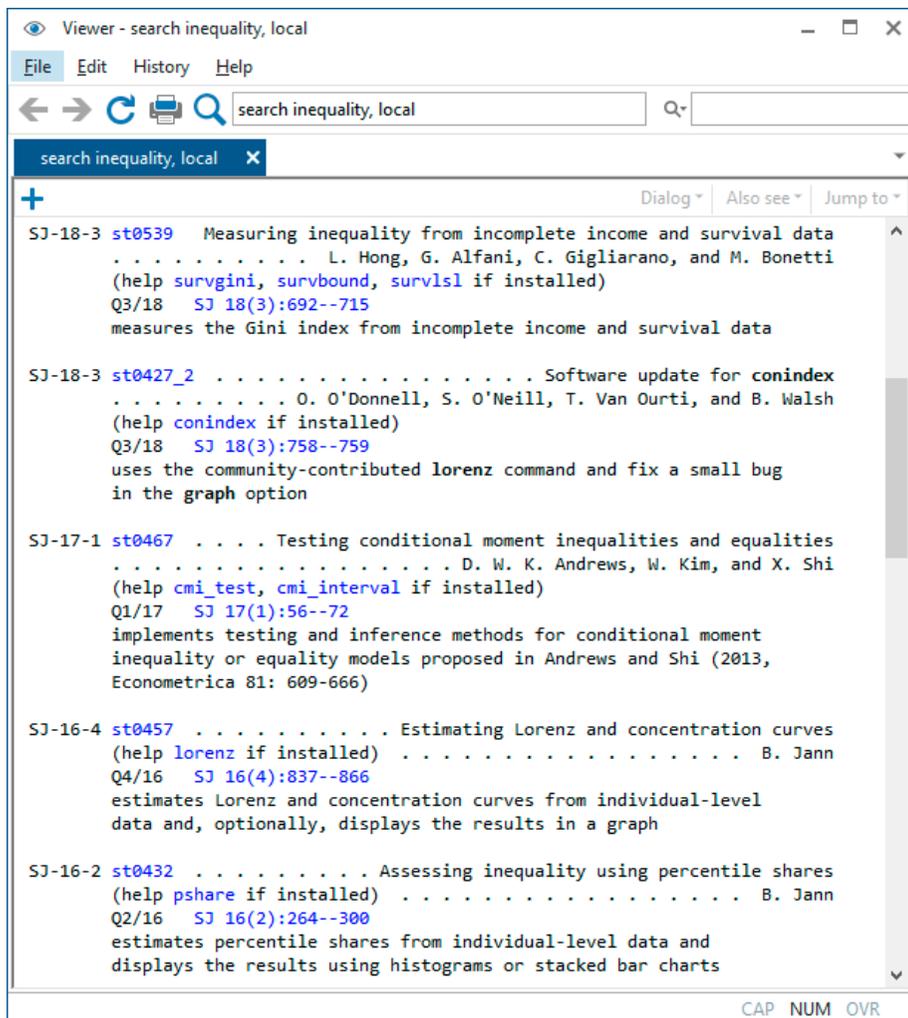
The Stata Journal

When searching in Stata, you will often see links to the *Stata Journal*.

The *Stata Journal* is a printed and electronic journal, published quarterly, containing articles about statistics, data analysis, teaching methods, and effective use of Stata's language. The *Journal* publishes peer-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 *Journal* 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 Stata, in particular. See https://www.stata-journal.com for more information.

Associated with each issue of the *Stata Journal* are the programs and datasets described therein. These programs and datasets are made available for download and installation over the Internet, not only to subscribers but also to all Stata users. See [R] **net** and [R] **sj** for more information.

Because the *Stata Journal* has had several articles about measures of inequality, if you select **Help > Search...**, choose *Search documentation and FAQs*, type inequality, and scroll down a bit, you will see some of these references:

SJ-18-3 refers to volume 18, number 3 of the *Stata Journal*. `st0539` refers to the package name; `st` indicates that this package is in the "statistics" category of the *Stata Journal*. Listed next is the title of the software package and the authors. The community-contributed commands found within this package are listed next in parentheses, followed by the reference details of the article. Clicking on an SJ link, such as `SJ 18(3):692--715`, will open a browser and take you to the *Stata Journal* website, where you can view the abstract of the article and purchase the article. The `search` listing concludes with a brief description of the community-contributed package.

The *Stata Journal* website allows all articles older than three years to be downloaded for free. See *Downloading community-contributed commands* in [GSW] **19 Updating and extending Stata— Internet functionality** for more details on how to install community-contributed software. Also see [R] **ssc** for information on a convenient interface to resources available from the Statistical Software Components (SSC) Archive.

We recommend that all users subscribe to the *Stata Journal*. See [U] **3.4 The Stata Journal** for more information.

Links to other sites where you can freely download programs and datasets for Stata can be found on the Stata website; see https://www.stata.com/links/.

5 Opening and saving Stata datasets

How to load your dataset from disk and save it to disk

Opening and saving datasets in Stata works similarly to those tasks in other computer applications. There are a few differences, however. First, it is possible to save and open files from within Stata's Command window. Second, Stata allows just one dataset to be active at any one time. That is, while it is possible to have multiple datasets in memory at once (see [D] **frames intro**), only one dataset may be active. Keeping this in mind will make Stata's care in opening new datasets clear. This chapter outlines all the possible ways to open and save datasets.

A Stata dataset can be opened in a variety of ways, most of which are probably familiar to you from other applications:

- Double-click on a Stata data file, which is a file whose extension is `.dta`. Note: The file extension may not be visible, depending on what options you have set in your operating system.
- Select **File > Open...** or click on the **Open** button and navigate to the file.
- Select **File > Open data subset...**, navigate to the file, specify the observation range, and select variables from the dataset.
- Select **File > Recent files >** *filename*.
- Type `use` *filename* in the Command window. Stata will look for *filename* in the current working directory. If the file is located elsewhere, you will need to give its path. Be aware that if there is a space anywhere in the path or filename, you will need to put the filename inside quotation marks. See [U] **11.6 Filenaming conventions**.
- Type `sysuse` *filename* in the Command window. Stata will look for *filename* in a series of directories called the `adopath`. Typically, this is for finding example datasets installed when you installed Stata, but it can also be used for easy access to your own datasets. For more information on the `adopath`, see [P] **sysdir**.
- Type `webuse` *filename* in the Command window. The `webuse` command is used to access datasets used in the Stata manuals; for example, `webuse lbw` loads the `lbw` dataset used in the documentation of the `logistic` command. For more information, see [D] **webuse**.

Opening a dataset in the current frame (see [D] **frames intro**) will replace the dataset, if any, that is currently in memory for that frame. Datasets in other frames are unaffected. If there have been changes to the data in the dataset in the current frame, Stata will refuse to discard the dataset unless you force it to do so. If you open the file with any method other than the Command window, you will be prompted. If you use the Command window and the current data have changed, you will get the following error message:

```
. sysuse auto
no; dataset in memory has changed since last saved
r(4);
```

These behaviors protect you from mistakenly losing data.

To save an unnamed dataset (or an old dataset under a new name):

1. select **File > Save as...**; or
2. type `save` *filename* in the Command window.

To save a dataset for use with Stata 13,
1. select **File > Save as...**, and select **Stata 13 Data (*.dta)** from the **Save as type** list; or
2. type `saveold` *filename* in the Command window.

To save a dataset that has been changed (overwriting the original data file),
1. select **File > Save**;
2. click on the **Save** button; or
3. type `save, replace` in the Command window.

Once you overwrite a dataset, there is no way to recover your original dataset. With important datasets, you may want to either keep a backup copy of your original *filename*`.dta` or save your changes to a dataset under a new name. This is no different from working with a word-processing document, except that recovering from an inadvertent save of a dataset is nearly impossible.

Important note: Changes you have made to a dataset are not permanent until you save them. You work with a copy of the dataset in memory, not with the data file itself. This should not be surprising, because it is the way that you work with almost all applications on your computer.

If you do not want to save your dataset, you can clear the dataset in memory and open a new dataset by typing `use` *filename*`, clear`.

6 Using the Data Editor

The Data Editor

The Data Editor gives a spreadsheet-like view of data that are currently in memory. You can use it to enter new data, edit existing data, search through the dataset, and edit attributes of the data in the dataset, such as variable names, labels, and display formats, as well as value labels.

In addition to the view of the data, there are two windows for manipulating variables and their properties: the Variables window and the Properties window. These are similar to the same-named windows in the main Stata window.

Any action you take in the Data Editor results in a command being issued to Stata as though you had typed it into the Command window. This means that you can keep good records and learn commands by using the Data Editor.

The Data Editor can be kept open while you work in Stata, giving you a live view of your dataset as you work. To protect your data from inadvertent changes, the Data Editor has two modes: edit mode for active editing and browse mode for viewing. In browse mode, editing within the Data Editor window is disabled. We highly recommend that you use the Data Editor in browse mode and switch to edit mode only when you want to make changes.

You can print your data from the Data Editor by selecting **Print** from the **File** menu.

We will be entering and editing data in this chapter, as well as manipulating the variables by using the Variables and Properties windows, so start the Data Editor in edit mode by clicking on the **Data Editor (Edit)** button, .

Buttons on the Data Editor

The toolbar for the Data Editor has some standard buttons and some buttons we have not yet seen:

 Edit mode: Changes the Data Editor to edit mode.

 Browse mode: Changes the Data Editor to browse mode for safely looking at data.

 Open: Opens a Stata dataset. Stata will warn you if your current dataset has unsaved changes.

 Save: Saves the dataset visible in the Data Editor.

 Print: Prints the dataset visible in the Data Editor.

 Copy: Copies the current selection to the Clipboard.

 Paste: Pastes the contents of the Clipboard. You may paste only if one cell is selected—this cell will become the upper-left corner of the pasted contents. Warning: This action will paste over existing data.

🔍 **Find**: Opens the Find bar for searching in the Data Editor.

▼ **Filter observations**: Filters the observations visible in the Data Editor. This button is useful for looking at a subset of the current dataset.

You can move about in the Data Editor by using the typical methods:

- To move to the right, use the *Tab* key or the right arrow key.
- To move to the left, use *Shift+Tab* or the left arrow key.
- To move down, use *Enter* or the down arrow key.
- To move up, use *Shift+Enter* or the up arrow key.

You can also click within a cell to select it.

Right-clicking within the Data Editor brings up a contextual menu that allows you to manipulate the data and what you are viewing. Right-clicking on the Data Editor window displays a menu from which you can do many common tasks:

- **Copy** to copy data to the Clipboard.
- **Paste** to paste data from the Clipboard.
- **Paste special...** to paste data from the Clipboard with finer control of delimiters, giving a preview of what will be pasted.
- **Select all** to select all the data displayed in the Data Editor. This could be different from the data in the dataset if the data are filtered or some variables are hidden.
- **Data** to open a submenu containing
    - **Insert variable...** to bring up a dialog for creating a new variable at the current cursor position.
    - **Add variable...** to bring up a dialog for creating a new variable at the beginning or end of a dataset.
    - **Replace contents of variable...** to bring up a dialog for replacing the values of the selected variable.
    - **Insert observations...** to bring up a dialog for inserting new empty observations at the current cursor position.
    - **Add observations...** to bring up a dialog for adding new empty observations to the end of the dataset.
    - **Sort data...** to sort the dataset by the selected variable.
    - **Value labels** to access a submenu for managing and displaying value labels.
    - **Keep only selected data** to keep only the selected data in the dataset. All remaining data will be dropped (removed) from the dataset. As always, this affects only the data in memory. It will not affect any data on disk.
    - **Drop selected data** to drop the selected data. This is only possible if the selection consists of either entire variables (columns) or observations (rows).
    - **Convert variables from string to numeric...** for converting string variables to numeric variables, which is useful when the string variables contain characters for formatting numbers instead of just numbers.
    - **Convert variables from numeric to string...** for converting numeric variables to strings.
    - **Encode string variable to labeled numeric...** for encoding a string-valued categorical variable to a numeric variable while still displaying the categories in tables and graphs.
    - **Decode labeled numeric variable to string...** for turning an encoded variable back into a string variable.
- **Reset selected column widths** to reset the selected columns to the default width.
- **Hide selected variables** to hide the selected variables.

- **Show only selected variables** to hide all but the selected variables.
- **Show entire dataset** to turn off all filters and unhide all variables.
- **Preferences...** to set the preferences for the Data Editor.
- **Font...** to change the font of the Data Editor.

Data entry

Entering data into the Data Editor is similar to entering data into a spreadsheet. One major difference is that the Data Editor has the concept of observations, which makes the data entry smart. We will illustrate this with an example. It will be useful for you to follow the example at your computer. To work along, you will need to start with an empty dataset, so save your dataset if necessary, and then type clear in the Command window.

Note: As a check to see if your data have changed, type describe, short (or d,s for short). Stata will tell you if your data have changed.

Suppose that we have the following data, and we want to enter them into Stata:

Make	Price	MPG	Weight	Gear ratio
VW Rabbit	4697	25	1930	3.78
Olds 98	8814	21	4060	2.41
Chev. Monza	3667		2750	2.73
AMC Concord	4099	22	2930	3.58
Datsun 510	5079	24	2280	3.54
	5189	20	3280	2.93
Datsun 810	8129	21	2750	3.55

We do not know MPG for the third car or the make of the sixth.

Start by opening the Data Editor in edit mode. You can do this either by clicking on the **Data Editor (Edit)** button, ⊞ , or by typing edit in the Command window. You should be greeted by a Data Editor with no data displayed. (If you see data, type clear in the Command window.) Stata shows the active cell by highlighting it and displaying *varname*[*obsnum*] next to the input box in the Cursor Location box. We will see below that we can navigate within a dataset by using this cell reference. The Data Editor starts, by default, in the first row of the first column. Because there are no data, there are no variable names, and so Stata shows var1[1] as the active cell.

We can enter these data either by working across the rows (observation by observation) or by working down the columns (variable by variable). To enter the data observation by observation, press *Tab* after entering each value until you have reached the end of the first row. In our case, we would type VW Rabbit, press *Tab*, type 4697, press *Tab*, and continue entering data to complete the first observation.

After you are finished with the first observation, select the second cell in the first column, either by clicking within it or by navigating to it. At this point, your screen should look like this:
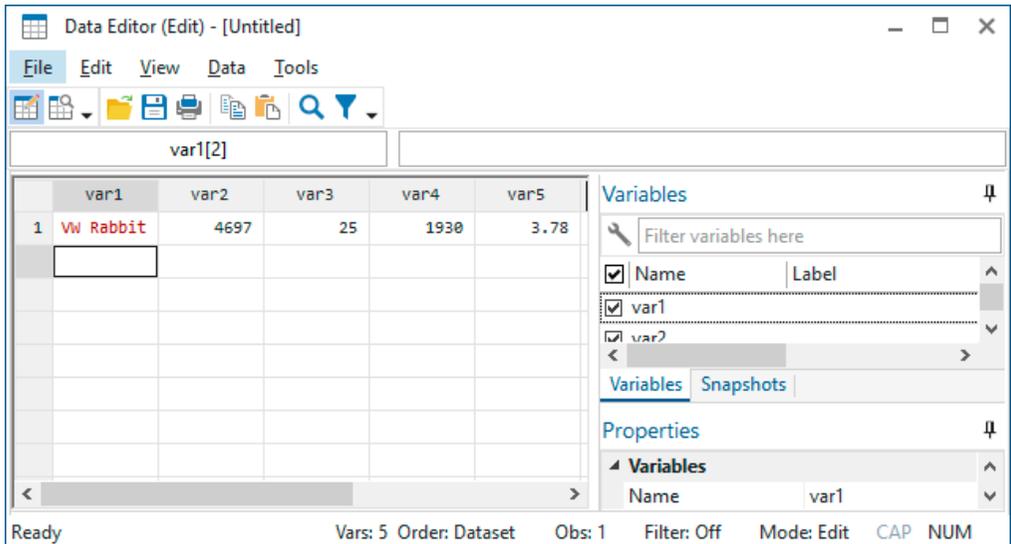
We can now enter the data for the second observation in the same fashion as the first—with one nice difference: after we enter the last value in the row, pressing the *Tab* key will bring us to the first cell in the third row. This is possible because the number of variables is known after the first observation has been entered, so Stata knows when it has all the data for an observation.

We can enter the rest of the data by pressing the *Tab* key between entries, simply skipping over missing values by tabbing through them.

If we had wanted to enter the data variable by variable, we could have done that by pressing *Enter* between each make of car until all seven observations were entered, skipping past the missing entry by pressing *Enter* twice. Once the first variable was entered, we would select the first cell in the second column and enter the price data. We would continue this until we were finished.

Notes on data entry

There are several things to note about data entry and the feedback you get from the Data Editor as you enter data:

- **Stata does not allow blank columns or rows in the middle of your dataset.**
  Whenever you enter new variables or observations, always begin in the first empty column or row. If you skip columns or rows, Stata will fill in the intervening columns or rows with missing values.

- **Strings and value labels are color coded.**
  To help distinguish between the different types of variables in the Data Editor, string values, value labels (see [GSW] **9 Labeling data**), and all other values are displayed in different colors. You can change the colors for strings and value labels by right-clicking on the Data Editor window and selecting **Preferences...**.

- **A period (.) represents Stata's system missing numeric value.**
  Stata has a system missing value, '.', and extended missing values '.a' through '.z'. By default, Stata uses its system missing value.

- **The *Tab* key is smart.**
  As we saw above, after the first observation has been entered, Stata knows how many variables you have. So at the end of the second observation (and all subsequent observations), *Tab* will automatically take you back to the first column.
- **The Cursor Location box both shows location and is used for navigation.**
  The Cursor Location box gives the location of the current cell. If you see, for example, `var3[4]`, this means that the current cell is the fourth observation of the variable named `var3`. You can navigate to a particular cell by typing the variable name and the observation in the Cursor Location box. If you wanted the second observation of `var1` to be the active cell, typing `var1 2` in the Cursor Location box and pressing *Enter* would take you there.
- **Double quotes around text are unnecessary in string variables.**
  Once Stata knows that a variable is a string variable (it holds text), there is no need to put quotes around the values, even if the values look like a number. Thus, if you wanted to enter ZIP codes as text, you would enter the first ZIP code with quotes (`"02173"`), but the rest would not need any quotes.
- **The arrow keys are context sensitive.**
  If you select a cell and type new data, using an arrow key will accept the change and move to a new active cell. If you double-click on a cell, you can edit within the cell contents. In this case, the right- and left-arrow keys move within the cell's data.
- **You can throw away changes to a cell.**
  If, while you are entering data in a cell, you decide you would like to cancel the changes, press the *Esc* key or click outside the cell.

Renaming and formatting variables

The data have now been entered into Stata, but the variable names leave something to be desired: they have the default names `var1`, `var2`, ..., `var5`. We would like to rename the variables so that they match the column titles from our dataset. We would also like to give the variables descriptions and change their formatting.

We will step through changing the name, label, and format of the `price` variable. We will then add a note to the variable. Start by clicking on the `var2` variable in the Variables window. The few properties associated with `var2` are now visible and editable in the Properties window. We may now systematically change the properties of `var2` to our choosing:

1. Double-click on `var2` in the *Name* field to select the old variable name, and type `price` to overwrite the name.
2. Click under the new `price` name in the *Label* field.
3. Enter a worthwhile label, such as `Price in dollars`.
4. Click on `%9.0g` in the *Format* field.
5. Click on the ellipsis (...) button that appears. The *Create format* dialog opens.
6. You can see here that there are many possible formats, most of which are related to time. We want commas in our numbers, so check the *Use commas in numeric output* checkbox. When you are done, click on the **OK** button.
7. Click in the *Notes* field.
8. Click on the ellipsis (...) button that appears. A dialog called *Notes for price* opens.
9. Click on the **Add** button and type a clever note.
10. When you are done typing, click on the **Submit** button, and then click on the **Close** button. This note is now attached to the `price` variable.
11. Click on the disclosure control to see the note you just typed in the Properties window.

To edit the properties of another variable, click on the variable in the Variables window. We can name the first variable `make`; the third, `mpg`; the fourth, `weight`; and the fifth, `gear_ratio`. Just before you rename `var5` to `gear_ratio`, your screen should look like this:
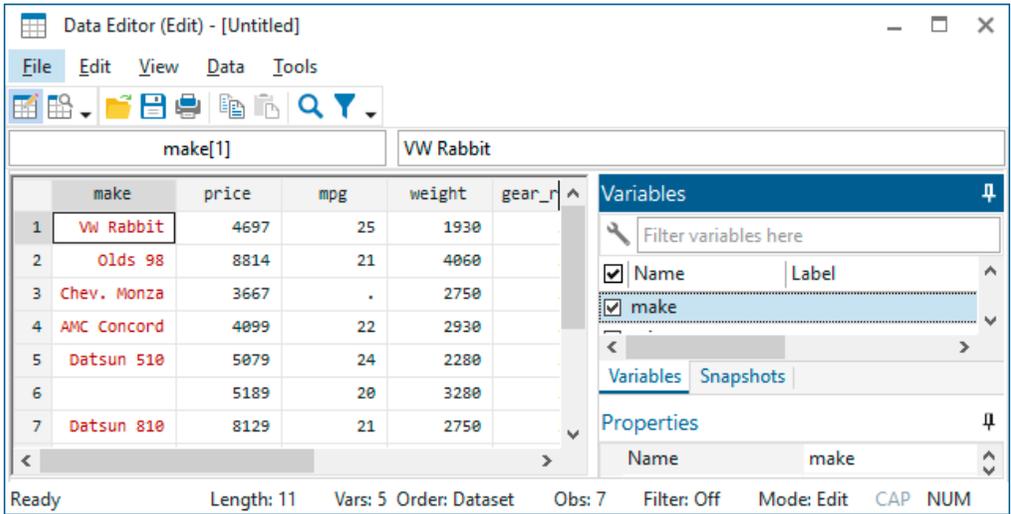
You need to know some rules for variable names:
- Stata is case sensitive.
  `Make`, `make`, and `MAKE` are all different names to Stata. If you had named your variables `Make`, `Price`, `MPG`, etc., then you would have to type them correctly capitalized in the future. Using all lowercase letters is easier.
- A variable name must be 1–32 characters long.
- The characters can be letters (`A–Z`, `a–z`), digits (`0–9`), underscores (`_`), or Unicode characters that are not symbols.
- Spaces or other characters are not allowed.
- The first character of a variable name must be a letter, an underscore, or a Unicode character. Although you can use an underscore to begin a variable name, it is highly discouraged. Such names are used for temporary variable names in Stata. You would like your data to be permanent, so using a temporary name could lead to great frustration.

For more information about variable names and value labels, see [GSW] **9 Labeling data**; for display formats, see [U] **12.5 Formats: Controlling how data are displayed**.

Copying and pasting data

You can copy and paste data by using the Data Editor. This is often a simple way to bring data into Stata from any other applications such as spreadsheets or databases.
1. Select the data that you wish to copy by using one of these means:
   - Click once on a variable name or column heading to select an entire column.
   - Click once on an observation number or row heading to select the entire row.
   - Click and drag the mouse to select a range of cells.
2. Copy the data to the Clipboard by right-clicking within the selected range, and select **Copy**.
3. Paste the data from the Clipboard by right-clicking on the top left cell of the area to which you wish to paste, and select **Paste**.

We will illustrate copying and pasting an observation by making a copy of the first observation and pasting it at the end of the dataset.

Start by clicking on the observation number of the first observation. Doing so highlights all the data in the row. Right-click on the same location (there is no need to move the mouse), and select **Copy**:
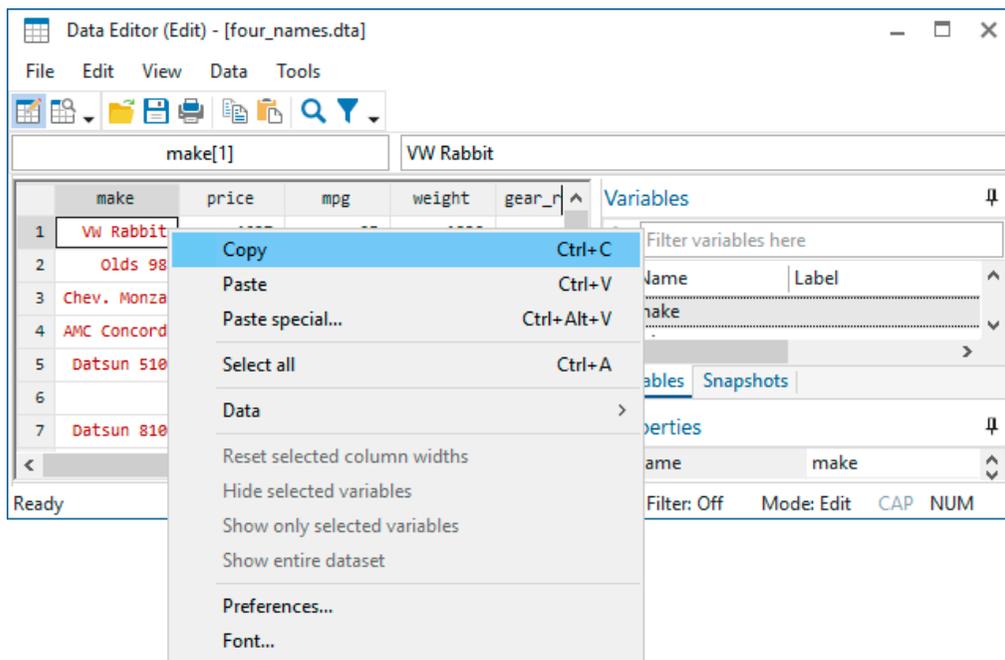
Click on the first cell in the eighth row, right-click while you are still in that cell, and choose **Paste** from the resulting menu. You can see that the observation was successfully duplicated.

Notes on copying and pasting

- The above example illustrated copying and pasting within the Data Editor. You can use roughly the same technique to copy and paste between other applications and Stata and between Stata and other applications. The easiest way to see if copying and pasting works properly with another application is to try it. The one requirement for things to work well is that the external application must copy tables in some delimited form, as do spreadsheet applications, many database applications, and some word processors. Using **Edit > Paste special...** gives some added flexibility to the formats you can paste into the Data Editor. If a simple paste does not give you what you expected, you should try **Edit > Paste special...**. For more information on file-based methods for importing data into Stata, see [GSW] **8 Importing data**.

- If you are copying and pasting data with value labels, you have a choice. You can copy variables with value labels as text, using the value labels as the actual values, or you can copy said variables as their underlying encoded numbers. Copying with the value labels is the default. If you would like the other choice, right-click in the Data Editor and select **Data > Value labels > Hide all value labels**, or select **Tools > Value labels > Hide all value labels**.

Changing data

As its name suggests, the Data Editor can be used to edit your dataset. As we have seen already, it can be used to edit the data themselves as well as the description and display options for the variables.

Here is an example for making some changes to the automobile dataset, which illustrates both methods for using the Data Editor and its documentation trail. We will also keep snapshots of the dataset as we are working so that we can revert to previous versions of the dataset in case we make a mistake.

We would like to investigate the dataset, work with value labels, delete the trunk variable, and make a new variable showing gas consumption per 100 miles. These tasks will illustrate the basics of working in the Data Editor.

Start by typing sysuse auto into the Command window. If you worked the previous example, you will get an error and are told that the dataset in memory has changed since it was last saved. This is good—Stata is keeping you from inadvertently throwing away the unsaved changes to your current dataset as it loads auto.dta. If you would like to save the dataset you have been working on, select **File > Save** and save the dataset in an appropriate location. Otherwise, type clear in the Command window, and press *Enter* to clear out the data, and then load that auto.dta.

Once auto.dta is loaded, start the Data Editor.

1. We remember that our grandfather had a Toronado, which looked sleek but seemed to require a lot of fill-ups. We would like to see if this car is in the dataset. To find it, we select **Edit > Find...**, type Toronado, and press *Enter*. We see that this make of car got 16 miles per gallon.

2. We would like to see which cars have the lowest and highest gas mileages. To do this, right-click on the column heading of the mpg column. Select **Data > Sort data...** from the contextual menu. A dialog will pop up asking how you want to sort, defaulting to sorting in ascending order. Click on **OK**. (Stata worries about sort order because sort order can affect reproducibility when using resampling techniques. This is a good thing.) You will see that the data have now been sorted by mpg in ascending order. The lowest-mileage cars are at the top of the screen; by scrolling to the bottom of the dataset, you can find the highest-mileage cars. You also could have sorted by selecting **Tools > Sort data...** once the mpg variable was selected.

3. We would like to investigate repair records and hence sort by the rep78 variable. (Do this now.) We see that the Starfire and Firebird both had poor repair records, but we would like to see the cars with good repair records. We could scroll to the bottom of the dataset, but it will be faster to use the Cursor Location box: type rep78 74 and press *Enter* to make rep78[74] the active cell. We notice that the last five entries for rep78 appear as dots. The dots mean that these values are missing. A few items of note:

   - As we can see from the result of the sort, Stata views missing values as being larger than all numeric nonmissing values. In technical terms, this means that rep78 >= . is equivalent to missing(rep78).
   - What we do not see here is that Stata has multiple missing-value indicators: . is Stata's default or system missing-value indicator, and .a, .b, ..., .z are Stata's extended missing values. Extended missing values are useful for indicating the reason why a value is unknown.
   - The different missing values sort among themselves: $. < .a < .b < \cdots < .z$. See [U] **12.2.1 Missing values** for full details.

4. We would like to make the repair records readable. Click on rep78 in the Variables window.

5. Click on the *Value label* field in the Properties window, and then click on the ellipsis (...) button that appears. This opens the *Manage value labels* dialog. We need to define a new value label for the repair records.

a. Click on the **Create label** button. You will see the *Create label* dialog.

b. Type a name for the label, say, `repairs`, in the *Label name* box.

c. Press the *Tab* key or click within the *Value* field.

d. Type 1 for the value, press the *Tab* key, and type `atrocious` for the label.

e. Press the *Enter* key to create the pairing.

f. Repeat steps d and e to make all the pairings: 2 with "bad", 3 with "OK", 4 with "good", and 5 with "stupendous".

g. Click on the **OK** button to finish creating the value label.

h. Click on the disclosure control, ⊞, to show the label—you should see this:
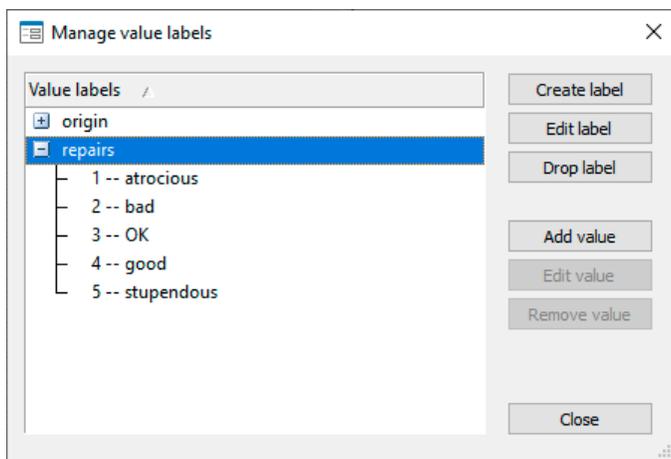
If you have something else, you can edit the label by clicking on the **Edit label** button.

i. Click on the **Close** button to close the *Manage value labels* dialog.

Now that the label has been created, attach it to the `rep78` variable by clicking on the down arrow in the *Value label* field and selecting the **repairs** label. You can now see the labels displayed in place of the values.

6. Suppose that we found the original source of the data in a time capsule, so we could replace some of the missing values for `rep78`. We could type the values into cells. We can also assign the values by right-clicking within a cell with a missing value and choosing a value from **Data > Value labels > Assign value from value label** "**repairs**". This strategy can be useful when a value label has many possible values.

7. We would now like to delete the `trunk` variable. We can do this by right-clicking on the `trunk` variable name at the top of the column and selecting the **Data > Drop selected data** menu item. Because this can lead to data loss, the Data Editor asks whether we would like to drop the selected variable. Click on the **Yes** button.

8. To finish up, we would like to create a variable containing the gallons of gasoline per 100 miles driven for each of the cars.

a. Right-click within any cell, and choose the **Data > Add variable...** menu item to bring up the *generate* dialog.

b. Type `gp100m` in the *Variable name* field.

c. Being sure that the *Specify a value or an expression* radio button is selected, type `100/mpg` in its field. We could have clicked on the **Create...** button to open the *Expression Builder* dialog, but this formula was simple enough to type. (You might want to explore the Expression Builder right now to see what it can do.)

    d. Be sure that the *Add at the end of dataset* item is chosen from the *Position of new variable* list.

    e. Click on **OK**. You can scroll to the right to see the newly created variable.

Throughout this data-editing session, we have been using the Data Editor to manipulate the data. If you look in the Results window, you will see the commands and their output. You can also see all the commands generated by the Data Editor in the History window. If you wanted to save the editing commands to use again later, you could do the following steps:

1. Click in the History window on the last command that came from the Data Editor.
2. Scroll up until you find the `sort mpg` command you ran immediately after opening the Data Editor, and *Shift*-click on it.
3. Right-click on one of the highlighted commands.
4. Select **Send selected to Do-file Editor**.

This procedure will save all the commands you highlighted into the Do-file Editor. You could then save them as a do-file, which you could run again later. We will talk more about the Do-file Editor in [GSW] **13 Using the Do-file Editor—automating Stata**. You can find help about do-files in [U] **16 Do-files**.

If you want to save this dataset, save it under a new name by using **File > Save as...** in the main Stata window to prevent overwriting the original dataset.

Working with snapshots

The Data Editor allows you to save to disk snapshots of whatever dataset you are working on. These are temporary copies of the dataset—they will be deleted when you exit Stata, so they need to be treated as temporary. Still, there are many uses for snapshots, such as

- saving a temporary copy of the data in memory so that another dataset can be opened and viewed;
- saving stages of work, which can be recovered in case you do something disastrous; and
- saving pieces of datasets while doing analyses.

We will keep using `auto.dta` from above; if you are starting here, you can start fresh by typing `sysuse auto` in the Command window to open the dataset. (If you get a warning about data in memory being lost, either use `clear` or save your data. See [GSW] **5 Opening and saving Stata datasets** for more information.) If we open the Data Editor and click on the **Snapshots** tab beneath the *Variables* window, we see the following window. If you are starting afresh, you will see numbers rather than labels for `rep78`.

	Data Editor (Edit) - [auto.dta]						
File	**Edit**	**View**	**Data**	**Tools**			

	make	price	mpg	rep78	headroom	∧	Snapshots
1	Pont. Firebird	4,934	18	atrocious	1.5		➕ ➖ ⬛ Restore
2	Olds Starfire	4,195	24	atrocious	2.0		# Label Timestamp Source
3	Cad. Eldorado	14,500	14	bad	3.5		There are no items to show.
4	Dodge Magnum	5,886	16	bad	4.0		
5	Dodge St. Regis	6,342	17	bad	4.5		
6	Plym. Volare	4,060	18	bad	5.0		Variables Snapshots
7	Dodge Diplomat	4,010	18	bad	4.0		Properties
8	Chev. Monte Carlo	5,104	22	bad	2.0		Type str18
9	Chev. Monza	3,667	24	bad	2.0	∨	Format %-18s
							Value label

Ready   Length: 18   Vars: 12 Order: Dataset   Obs: 74   Filter: Off   Mode: Edit   CAP NUM

To begin with, only one button is active in the Snapshots toolbar. Click on the active button—the **Add** button, ➕ . It brings up a dialog asking for a label, or name, for the snapshot. Give it an inventive name, such as `Start`, and press *Enter*. You can see that a snapshot is now listed in the Snapshots window, and all the buttons in the toolbar are now active. The following buttons appear in the Snapshots window:

➕ **Add**: Save a new snapshot with a timestamp and label.

➖ **Remove**: Erase a snapshot. This action deletes the temporary snapshot file but does not affect the data in memory.

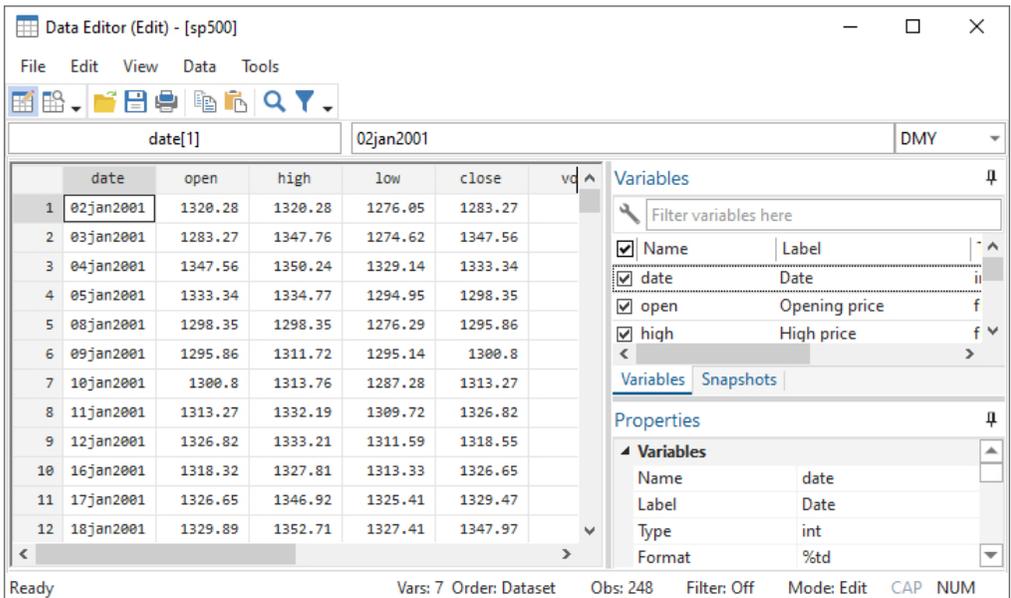⬛ **Change label**: Edit the label of the selected (highlighted) snapshot.

Restore **Restore**: Replace the data in memory with the data from the selected snapshot. You will get a dialog asking you to confirm your action.

You should now try manipulating the dataset by using the tools we have seen. Once you have done that, create another snapshot, calling it `Changed`. Open the Snapshots window and restore the `Start` snapshot by either double-clicking it or clicking first on it and then on the **Restore** button to see where you started. You can then go back to where you were working by restoring your `Changed` snapshot.

Snapshots continue to be available either until they are deleted or until you exit Stata. You can thus use snapshots of one dataset while working on another. You will find your own uses for snapshots—just take care to save the datasets you want for future use because the snapshots are temporary.

Dates and the Data Editor

The Data Editor has two special tools for working with dates in Stata. To see these in action, we will need to open another dataset. Either save your dataset or `clear` it out, and then type `sysuse sp500` in the Command window. Look in the Data Editor to see what you have.

You can see a `date` variable that has January 2, 2001, as its first day, though it is being displayed in Stata's default format for dates.

We will start with formatting:

1. Select the `date` variable in the Variables window to the right of the data table.
2. In the Properties window, select the Format row and click on the ellipsis button that appears.
3. The *Create format* dialog tells us three pieces of information about the date format:
    - These are daily dates. As you can see, Stata understands other types of dates that are often used in financial data.
    - Looking at the bottom of the dialog, you can see that Stata's default date format is `%td`. This means that the variable contains time values that are to be interpreted as daily dates.
    - This default format is displayed as, for example, `07apr2021`.
4. There are many premade date formats in the *Samples* pane at the top right of the *Create format* dialog. Click on `April 7, 2021`. You can see how the format would be specified at the bottom of the dialog.
5. Click on **OK** to close the *Create format* dialog. You can see that the dates are now displayed differently.

This is a very simple way to change date formats. For complete information on dates and date formats, see [D] **Datetime**.

We will now change some of the dates to illustrate how this can be done simply, regardless of the format in which the dates are displayed. If you look in the upper-right corner of the Data Editor, you will see the *Time/Date input mask* field, which shows `DMY`. This field affects how dates are entered when editing data.

By default, the input mask is set to DMY. This means dates can be entered in many different fashions, as long as the order of the date components is day, month, year. Try the following:

1. Click in the first observation of date so that the Cursor Location shows date[1].
2. Type 18jan2021 and press the *Enter* key. Stata understands the DMY input mask and knows enough to enter the new date in the selected cell.
3. Enter 30042021 and press *Enter*. Stata still understands the input mask, even though there are no separators.
4. Click within the *Time/Date input mask* field, and choose MDY from the drop-down menu.
5. Click on any observation in the date column.
6. Type March 15, 2021 and press *Enter*. Stata will still understand.

Working in this fashion is the fastest way to edit dates by hand. If you look in the Results window, you will see why.

We are now finished with this dataset, so type clear and press *Enter*.

Data Editor advice

As you could see above, a small mistake in the Data Editor could cause large problems in your dataset. You really must take care in how you edit your data.

- People who care about data integrity know that editors are dangerous—it is easy to accidentally make changes. **Never use the Data Editor in edit mode when you just want to look at your data.** Use the Data Editor in browse mode (or use the browse command).
- If you must edit your data, protect yourself by limiting the dataset's exposure. For example, if you need to change rep78 only if it is missing, find a way to look at just the missing values for rep78 and any other variables needed to make the change. This will make it impossible for you to change (damage) variables or observations other than those you view. We will explore this aspect shortly.
- Even with these caveats, Stata's Data Editor is safer than most because it records commands in the Results window. Use this feature to log your output and make a permanent record of the changes. Then you can verify that the changes you made are the changes you wanted to make. See [GSW] **16 Saving and printing results by using logs** for information on creating log files.

Filtering and hiding

We would now like to investigate restricting our view of the data we see in the editor. This feature is useful for the reasons mentioned above, and as we will see, it helps if we would like to browse through the data of a large dataset. In any case, we would like to focus on some data, not all the data, whether we focus on some of the variables, some of the observations, or even just some observations within some variables. We would also like to change the order of the variables. We will show you how this is done by using both the graphical interface and commands.

Open the automobile dataset by typing sysuse auto. If you get an error message, type clear and try again. Once you have done that, open the Data Editor.

Suppose that we would like to edit only those observations for which rep78 is missing. We will need to look at the make of the car so that we know which observations we are working with, but we do not need to see any other variables. We will work as though we had a very large dataset to work with.

1. Before we get started, try experimenting with the Variables window.
    a. Drag variables up and down the list. Doing so changes the order of the variables' columns in the Data Editor. It does not change their order in the dataset itself.
    b. Uncheck some of the checkboxes in the first column to hide some of the variables.
    c. Type a search criterion in the *Filter variables here* field. Just like in the Variables window in the main Stata window, the default is to ignore case and find any variables or variable labels containing any of the words in the filter. Clicking on the wrench on the left will allow you to change this behavior as well as to add or remove additional columns containing information about the variables. The filtering of variables in the list affects what is displayed in the Variables window; it does not affect what variables' data are displayed. When you are done, delete your filter text.
2. Right-click on any variable in the Variables window, and select **Select all** from the contextual menu.
3. Click on any checkbox to deselect all the variables.
4. Click on the make variable to select it, and deselect all the other variables.
5. Click on the checkbox for make.
6. Click on the checkbox for rep78.

If you look in the Command window, you can see that no commands have been issued, because hiding the variables does not affect the dataset—it affects only what shows in the Data Editor.

We now have protected ourselves by using only those variables that we need. We should now reduce our view to only those observations for which rep78 is missing. This is simple.

1. Click on the **Filter observations** button, ▼ , in the Data Editor's toolbar.
2. Enter missing(rep78) in the *Filter by expression* field.
3. Click on the **Apply filter** button.
4. If you are curious, click on the ellipsis button. It opens up an *Expression Builder* dialog. This lists the wide variety of functions available in Stata. See the *Stata Functions Reference Manual*.

Now we are focused on the part of the dataset in which we would like to work, and we cannot destroy or mistakenly alter other data by stray keystrokes in the Data Editor window.

It is worth learning how to hide variables and filter observations in the Data Editor from the Command window. This can be quite convenient if you are going to restrict your view, as we did above. To work from the Command window, we must use the edit command together with a *varlist* (variable list) along with if and in qualifiers in the Command window. By using a *varlist*, we restrict the variables we look at, whereas the if and in qualifiers restrict the observations we see. ([GSW] **10 Listing data and basic command syntax** contains many examples of using a command with a variable list and if and in.) Suppose we want to correct the missing values for rep78. The minimum amount of data we need to expose are make and rep78. To see this minimal amount of information and hence to minimize our exposure to making mistakes, we enter the commands

```
. sysuse auto
(1978 automobile data)
. edit make rep78 if missing(rep78)
```

and we would see the following window:
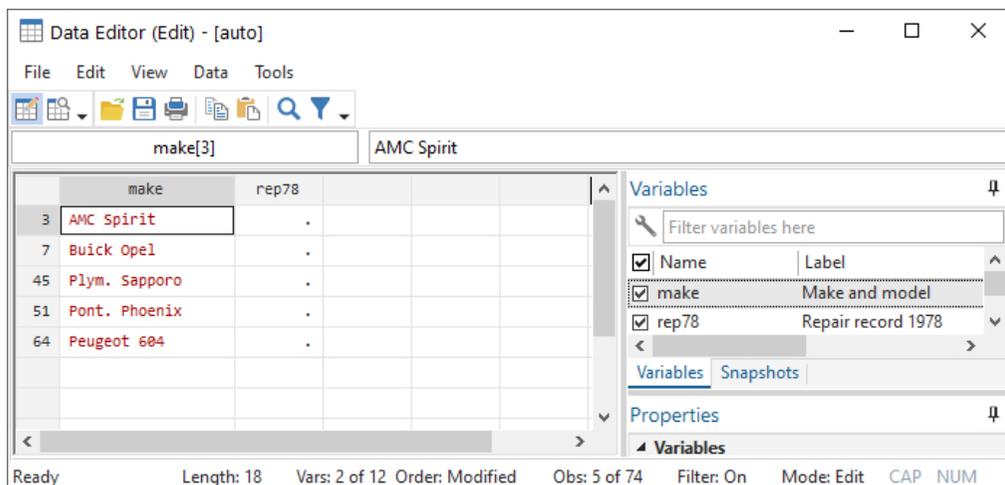
Once again, we are safe and sound.

Keep this lesson in mind if you edit your data. It is a lesson well learned.

Browse mode

The purpose of using the Data Editor in browse mode is to look at data without altering them by stray keystrokes. You can start the Data Editor in browse mode by clicking on the **Data Editor**

(Browse) button, , or by typing browse in the Command window. When you work in browse mode, all contextual menu items that would let you alter the data, the labels, or any of the display formats for the variables are disabled. You may view a variable's properties with the **Properties** menu item, but you may not make any changes. You still can filter observations and hide variables to get a restricted view because these actions do not change the dataset.

Note: Because you can still use Stata menus not related to the Data Editor and because you can still type commands in the Commands window, it is possible to change the data even if the Data Editor is in browse mode. In fact, this means you can watch how your commands affect the dataset. You are merely restricted from using the Data Editor itself to change the data.

7 Using the Variables Manager

The Variables Manager

This chapter discusses Stata's Variables Manager. To get started, open the automobile dataset by typing `sysuse auto, clear` in the Command window. You open the Variables Manager by selecting **Data > Variables Manager** or clicking on the **Variables Manager** button ⊞ .
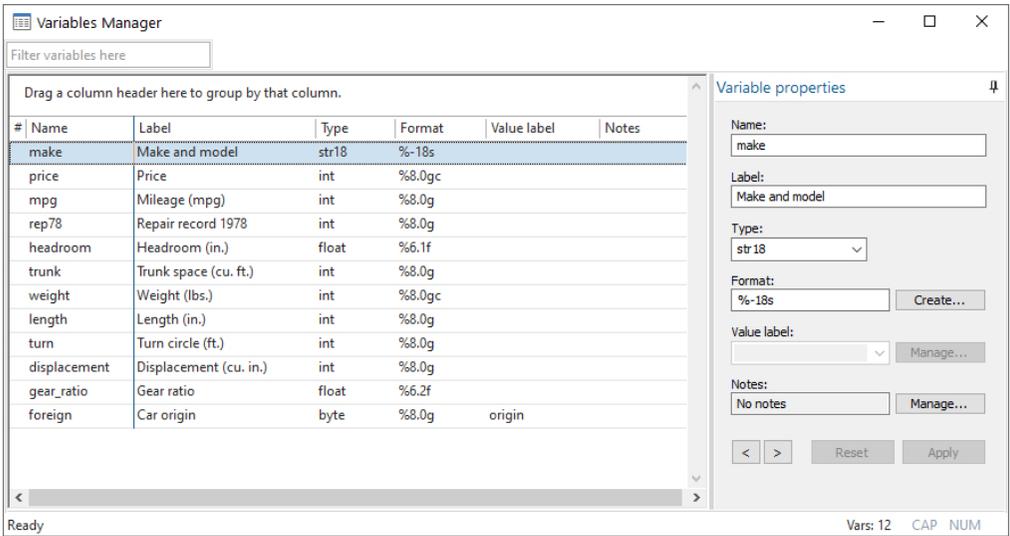
The Variables Manager is a tool for managing properties of variables both individually and in groups. It can be used to create variable and value labels, rename variables, change display formats, and manage notes. It has the ability to filter and group variables as well as to create variable lists. Users will find these features useful for managing large datasets.

Any action you take in the Variables Manager results in a command being issued to Stata as though you had typed it in the Command window. This means that you can keep good records and learn commands by using the Variables Manager.

The Variable pane

The left pane of the Variables Manager is called the Variable pane, though it has no explicit title on the screen. It shows the list of variables in the dataset. This list can be manipulated in a variety of ways.

- The variables can be filtered by entering text into the filter box in the upper-left corner. This can be a good way to zoom in on similarly named or labeled variables.
- The list can be sorted by clicking on the column title.
    a. If you click on a column title, it will sort in ascending order.
    b. A second click on the same column title will change to sorting in descending order.

c. Clicking on the hash mark (#) restores the sort order to the original variable order.

The sort order affects only how the data appear in the Variable Managers window—the dataset itself stays the same.

- The order of the columns can be changed by dragging the column titles. To restore the original column headings, right-click on the column titles and select **Restore column defaults**.
- The variables can be grouped by values in one or more columns. This is done by dragging the column titles into the grouping bar. The grouping can be canceled by dragging the column titles back into the column titles row. Here is an example of `auto.dta` grouped by variable type:

Grouping by variable label can be a good way to find unlabeled variables.

Right-clicking on the Variable pane

Right-clicking on the Variable pane displays a menu from which you can do many common tasks:

- **Edit variable properties** to change the focus to the Variable Properties pane. This will expose the Variable Properties pane if it has been automatically hidden.
- **Keep only selected variables** to keep only the selected variables in the dataset and to drop all the others.
- **Drop selected variables** to drop all the selected variables from the dataset.
- **Manage notes for selected variable...** to open a window that allows adding and deleting notes for a single variable. This is disabled if multiple variables are selected.
- **Manage notes for dataset...** to open a window that allows adding and deleting notes for the dataset as a whole.
- **Copy varlist** to copy the names of the selected variables to the Clipboard.
- **Select all** to select all visible variables. If a variable has become hidden because of the filter, it will not be selected.

- **Send varlist to Command window** to insert the names of the selected variables in the Command window. Combined with grouping and sorting, this can be a useful way to create variable lists in large datasets.
- **Print...** to print the Variable pane. You can change the widths of the printed columns by changing the widths of the columns in the Variables Manager.

The Variable properties pane

The Variable Properties pane can be used to manipulate the properties of variables selected in the Variable pane. With one variable selected, you can manipulate all properties of the variable. With many variables selected, you can change their formats or types as well as assign value labels all at once. These fields work in the same fashion as those shown in *Renaming and formatting variables* in [GSW] **6 Using the Data Editor**. We can also manage the notes Stata allows you to attach to variables and the dataset—we will show an example below.

The Variable Properties pane is, in actuality, a docking window, like those discussed in *Auto Hide and pinning* in [GSW] **2 The Stata user interface**. You can see this because of the pushpin in its upper-right corner. If you click on the pushpin, the window will automatically hide when not in use. You can also dock the window in another part of the Variables Manager window by dragging it by its title bar to one of the docking guides.

Managing notes

Stata allows you to attach notes to both variables and the dataset as a whole. These are simple text notes that you can use to document whatever you like—the source of the dataset, data collection quirks associated with a variable, what you need to investigate about a variable, or anything else.

Start by selecting a variable in the Variable pane. We will work with the `price` variable. Click on the **Manage...** button next to the *Notes* field, and you will see the following dialog appear:
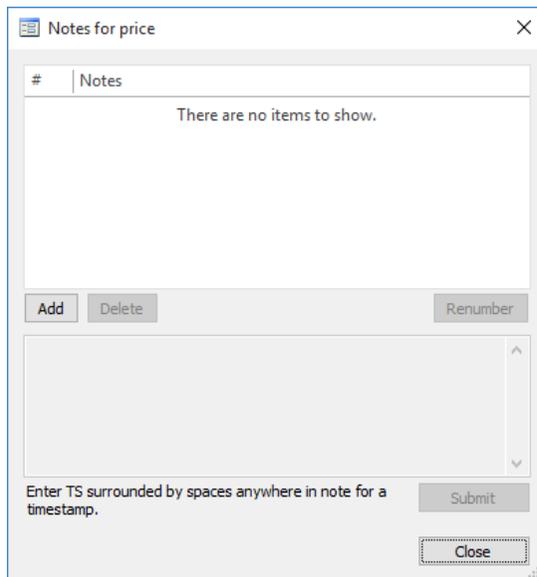
We will add a few notes:

1. Click on the **Add** button to add a note.
2. Type `TS - started working`. TS with a trailing space inserts a timestamp in the note.
3. Add two more notes. We added two notes about prices:
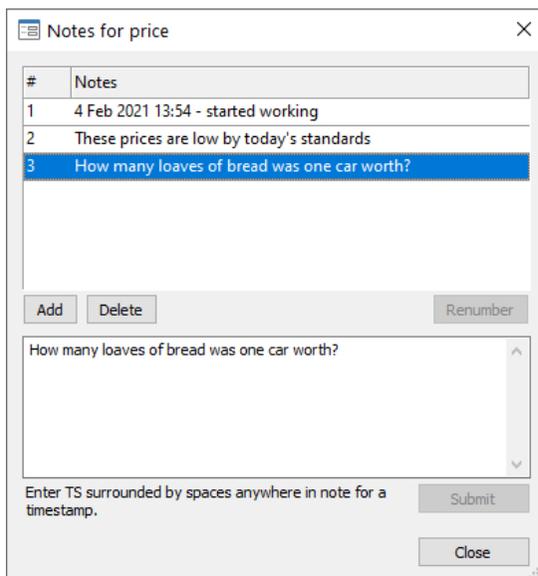
It is worth experimenting with adding, deleting, and editing notes. Notes can be an invaluable memory aid when working on projects that last a long time. Anytime you manipulate notes in the Notes Manager, you create Stata commands.

8   Importing data

Copying and pasting

One of the easiest ways to get data into Stata is often overlooked: you can copy data from most applications that understand the concept of a table and then paste the data into the Data Editor. This approach works for all spreadsheet applications, many database applications, some word-processing applications, and even some webpages. Just copy the full range of data, paste it into the Data Editor, and everything will probably work well. You can even copy a text file that has the pieces of data separated by commas and then paste it into the Data Editor.

Suppose that your friend has a small dataset about some very old cars.

```
VW Rabbit,4697,25,1930,3.78
Olds 98,8814,21,4060,2.41
Chev. Monza,3667,,2750,2.73
,4099,22,2930,3.58
Datsun 510,5079,24,2280,3.54
Buick Regal,5189,20,3280,2.93
Datsun 810,8129,,2750,3.55
```

You would like to put these data into Stata. Doing so is easier than you think:
1. Clear out your current dataset by typing clear.
2. Copy the above data.
3. Open the Data Editor in edit mode.
4. Select **Edit > Paste special...**.
5. Stata sees that the column delimiters are commas and shows how the data would look.
6. Click on the **OK** button.
You can see that Stata has imported the data nicely.

Later in this chapter, we would like to bring these data into Stata without copying and pasting, so we would like to save them as a text file. Go back to the main Stata window, and click on the **Do-file Editor** button, 📝 , to open a new Do-file Editor window. Paste the data in the Do-file Editor, then click on the **Save** button. Navigate to your working directory, and save the file as a few cars.csv. If you do not know what your working directory is, look in the status bar at the bottom of the main Stata window.

Be careful if you are copying data from a spreadsheet because spreadsheets can contain special formatting that ruins its rectangular form. Be sure that your spreadsheet does not contain blank rows, blank columns, repeated headers, or merged cells because these can cause trouble. As long as your spreadsheet looks like a table, you will be fine.

Commands for importing data

Copying and pasting is a great way to bring data into Stata, but if you need a clear audit trail for your data, you will need another way to bring data into Stata. The rest of this chapter will explain how to do this. You will also learn methods that lend themselves better to repetitive tasks and methods for importing data from a wide variety of sources.

Stata has various commands for importing data. The three main commands for reading non–Stata datasets in text are

- `import delimited`, which is made for reading text files created by spreadsheet or database programs or, more generally, for reading text files with clearly defined column delimiters such as commas, tabs, semicolons, or spaces;
- `infile`, which is made for reading simple data that are separated by spaces or rigidly formatted data aligned in columns; and
- `infix`, which is made for data aligned in columns but possibly split across rows.

Stata has other commands that can read other types of files and can even get data from external databases without the need for an interim file:

- The `import excel` command can read Microsoft Excel files directly, either as an `.xls` or as an `.xlsx` file.
- The `import sas` command can read native SAS files, so data can be transferred from SAS to Stata in this fashion.
- The `import spss` command can read IBM SPSS Statistics files.
- The `import sasxport5` command can read version SAS V5 Transport files. The `import sasxport8` command can read version SAS V8 Transport files.
- The `odbc` command can be used to pull data directly from any data sources for which you have ODBC (Open Database Connectivity) drivers.
- The `jdbc` command allows you to load data from a database, execute SQL statements on a database, and insert data into a database using JDBC (Java Database Connectivity) drivers.

Stata can import more formats; see [D] **import** for the full list.

Each command expects the file that it is reading to be in a specific format. This chapter will explain some of those formats and give some examples. For the full description, consult the *Data Management Reference Manual*.

The import delimited command

The `import delimited` command was developed to read in text files that were created by spreadsheet or database programs because these are common formats for sharing datasets on the Internet. All spreadsheet programs and most database applications have an option to save the dataset as a text file with the columns delimited with either tab characters or commas. Some of these programs also save the column titles (variable names, in Stata) in the text file.

To read in such a file, you have only to type `import delimited` *filename*, where *filename* is the name of the text file. The `import delimited` command will figure out what the delimiter character is (tab or comma) and what type of data is in each column. As always, if *filename* contains spaces, put double quotes around the filename, and include the path if *filename* is not in the current working directory.

By default, the `import delimited` command understands files that use the tab or comma as the column delimiter automatically. If you have a file that uses another character as the delimiter, use `import delimited`'s `delimiters()` option.

Earlier in this chapter, you saved a file called **a few cars.csv** in *Copying and pasting*. These data correspond to the make, price, MPG, weight, and gear ratio of a few very old cars. The variable names are not in the file (so `import delimited` will assign its own names), and the fields are separated by commas. Clear out any existing data, then use `import delimited` to read the data in this file. Because there are spaces in the filename, it must be enclosed in double quotes.

```
. clear
. import delimited "a few cars.csv"
(5 vars, 7 obs)
```

You can look at the data in the Data Editor, and it will look just like the earlier result from copying and pasting. We will now list the data so that we can see them in the manual. The `separator(0)` option suppresses the horizontal separator line that is drawn after every fifth observation by default.

```
. list, separator(0)

             v1      v2    v3      v4      v5

  1.    VW Rabbit    4697    25    1930    3.78
  2.      Olds 98    8814    21    4060    2.41
  3.   Chev. Monza    3667     .    2750    2.73
  4.                4099    22    2930    3.58
  5.    Datsun 510    5079    24    2280    3.54
  6.   Buick Regal    5189    20    3280    2.93
  7.   Datsun 810    8129     .    2750    3.55
```

If you want to specify better variable names, you can include the desired names in the command. When you specify variable names, you must also use the `using` keyword before the filename.

```
. import delimited make price mpg weight gear_ratio using "a few cars.csv"
(5 vars, 7 obs)
. list, separator(0)

             make    price   mpg   weight   gear_r~o

  1.    VW Rabbit     4697    25     1930     3.78
  2.      Olds 98     8814    21     4060     2.41
  3.   Chev. Monza    3667     .     2750     2.73
  4.                 4099    22     2930     3.58
  5.    Datsun 510    5079    24     2280     3.54
  6.   Buick Regal    5189    20     3280     2.93
  7.   Datsun 810    8129     .     2750     3.55
```

As a side note about displaying data, Stata listed `gear_ratio` as `gear_r~o` in the output from `list`. `gear_r~o` is a unique abbreviation for the variable `gear_ratio`. Stata displays the abbreviated variable name when variable names are longer than eight characters.

To prevent Stata from abbreviating `gear_ratio`, you could specify the `abbreviate(10)` option:

```
. list, separator(0) abbreviate(10)

              make    price    mpg    weight    gear_ratio

  1.     VW Rabbit     4697     25      1930          3.78
  2.       Olds 98     8814     21      4060          2.41
  3.   Chev. Monza     3667      .      2750          2.73
  4.                   4099     22      2930          3.58
  5.    Datsun 510     5079     24      2280          3.54
  6.   Buick Regal     5189     20      3280          2.93
  7.    Datsun 810     8129      .      2750          3.55
```

For more information on the ~ abbreviation and on `list`, see [GSW] **10 Listing data and basic command syntax**.

We will use this dataset again in the next chapter, so we would like to save it. Type `save afewcars`, and press *Enter* in the Command window to save the dataset.

For this simple example, you could have copied the contents of the file and pasted it into the Data Editor by using **Paste special...** and choosing comma as the delimiter.

For text files that have no nice delimiters or for which observations could be spread out across many lines, Stata has two more commands: `infile` and `infix`. See [D] **import** for more information about how to read in such files.

Importing files from other software

Stata has some more specialized methods for reading data that were created by other applications and stored in their proprietary formats.

The `import excel` command is made for reading files created by Microsoft Excel. See [D] **import excel** for full details.

The `import spss` command is made for reading files created by IBM SPSS Statistics. See [D] **import spss** for full details.

The `import sas` command is made for reading files created by SAS. See [D] **import sas** for full details.

The `import sasxport5` and `import sasxport8` commands can read SAS V5 and SAS V8 Transport files. See [D] **import sasxport5** and [D] **import sasxport8** for full details.

If you have software that supports ODBC, you can read data by using the `odbc` command without the need to create interim files. See [D] **odbc** for full details.

The `jdbc` command allows you to connect to, load data from, insert data into, and execute queries on a database using JDBC. See [D] **jdbc** for full details.

Here is a brief summary of the choices:
- If you have a Microsoft Excel .xls or .xlsx file, use `import excel`.
- If you have an IBM SPSS Statistics .sav file, use `import spss`.
- If you have a SAS .sas7bdat file created on a Windows machine, use `import sas`.

- If you have a file exported from a spreadsheet or database application to a tab-delimited or CSV file, use `import delimited`.
- If you have a fixed-format file, either use `infile` with a dictionary or use `infix`.
- If you have a database accessible with ODBC, use `odbc`.
- If you have a database accessible with JDBC, use `jdbc`.
- If you have a SAS V5 Transport file, use `import sasxport5`.
- If you have a SAS V8 Transport file, use `import sasxport8`.
- If you have economic data from the Federal Reserve Data, use `import fred`.
- If you subscribe to any Haver Analytics databases, use `import haver`.
- If you have a dBASE file, use `import dbase`.
- If you have a table, you could try copying it and pasting it into the Data Editor.
- Finally, you can purchase a third-party transfer program that will convert the other software's data file format to Stata's data file format.

9 Labeling data

Making data readable

This chapter discusses, in brief, labeling of the dataset, variables, and values. Such labeling is critical to careful use of data. Labeling variables with descriptive names clarifies their meanings. Labeling values of numerical categorical variables ensures that the real-world meanings of the encodings are not forgotten. These points are crucial when sharing data with others, including your future self. Labels are also used in the output of most Stata commands, so proper labeling of the dataset will produce much more readable results. We will work through an example of properly labeling a dataset, its variables, and the values of one encoded variable.

The dataset structure: The describe command

At the end of *The import delimited command* in [GSW] **8 Importing data**, we saved a dataset called `afewcars.dta`. We will put this dataset into a shape that a colleague would understand. Let's see what it contains.

```
. use afewcars
. list, separator(0)

              make    price   mpg   weight   gear_r~o

  1.      VW Rabbit    4697    25     1930       3.78
  2.        Olds 98    8814    21     4060       2.41
  3.    Chev. Monza    3667     .     2750       2.73
  4.                   4099    22     2930       3.58
  5.      Datsun 510    5079    24     2280       3.54
  6.     Buick Regal    5189    20     3280       2.93
  7.     Datsun 810    8129     .     2750       3.55
```

The data allow us to make some guesses at the values in the dataset, but, for example, we do not know the units in which the price or weight is measured, and the term "mpg" could be confusing for people outside the United States. Perhaps we can learn something from the description of the dataset. Stata has the aptly named `describe` command for this purpose (as we saw in [GSW] **1 Introducing Stata—sample session**).

```
. describe
Contains data from afewcars.dta
 Observations:            7
    Variables:            5                      21 Jan 2021 16:24

Variable      Storage   Display    Value
    name         type    format    label     Variable label

make           str18   %18s
price          float   %9.0g
mpg            float   %9.0g
weight         float   %9.0g
gear_ratio     float   %9.0g

Sorted by:
```

Though there is precious little information that could help us as a researcher, we can glean some information here about how Stata thinks of the data from the first three columns of the output.

1. The *Variable name* is the name we use to tell Stata about a variable.

2. The *Storage type* (otherwise known as the *data type*) is the way in which Stata stores the data in a variable. There are six different storage types, each having its own memory requirement:

   a. For integers:

      byte for integers between -127 and 100 (using 1 byte of memory per observation)

      int for integers between $-32{,}767$ and $32{,}740$ (using 2 bytes of memory per observation)

      long for integers between $-2{,}147{,}483{,}647$ and $2{,}147{,}483{,}620$ (using 4 bytes of memory per observation)

   b. For real numbers:

      float for real numbers with 8.5 digits of precision (using 4 bytes of memory per observation)

      double for real numbers with 16.5 digits of precision (using 8 bytes of memory per observation)

   c. For strings (text) between 1 and 2,045 bytes (using 1 byte of memory per observation per character for ASCII and up to 4 bytes of memory per Unicode character):

      str1 for 1-byte-long strings

      str2 for 2-byte-long strings

      str3 for 3-byte-long strings

      . . .

      str2045 for 2,045-byte-long strings

   d. Stata also has a strL storage type for strings of arbitrary length up to 2,000,000,000 bytes. strLs can also hold binary data, often referred to as BLOBs, or binary large objects, in databases. We will not illustrate these here.

   Storage types affect both the precision of computations and the size of datasets. A quick guide to storage types is available at help data types or in [D] **Data types**.

3. The *Display format* controls how the variable is displayed; see [U] **12.5 Formats: Controlling how data are displayed**. By default, Stata sets it to something reasonable given the storage type.

   We would like to make this dataset into something containing all the information we need.

To see what a well-labeled dataset looks like, we can look at a dataset stored at the Stata Press repository. We need not load the data (and disturb what we are doing); we do not even need a copy of the dataset on our machine. (You will learn more about Stata's Internet capabilities in [GSW] **19 Updating and extending Stata—Internet functionality**.) All we need to do is direct `describe` to look at the proper file by using the command `describe using` *filename*.

```
. describe using https://www.stata-press.com/data/r17/auto
Contains data                                    1978 automobile data
 Observations:              74                    13 Apr 2020 17:45
    Variables:              12

Variable      Storage   Display    Value
    name         type    format    label      Variable label

make           str18    %-18s                  Make and model
price          int      %8.0gc                 Price
mpg            int      %8.0g                  Mileage (mpg)
rep78          int      %8.0g                  Repair record 1978
headroom       float    %6.1f                  Headroom (in.)
trunk          int      %8.0g                  Trunk space (cu. ft.)
weight         int      %8.0gc                 Weight (lbs.)
length         int      %8.0g                  Length (in.)
turn           int      %8.0g                  Turn circle (ft.)
displacement   int      %8.0g                  Displacement (cu. in.)
gear_ratio     float    %6.2f                  Gear ratio
foreign        byte     %8.0g     origin       Car origin

Sorted by: foreign
```

This output is much more informative. There are three locations where labels are attached that help explain what the dataset contains:

1. In the first line, 1978 automobile data is the *data label*. It gives information about the contents of the dataset. Data can be labeled by selecting **Data > Data utilities > Label utilities > Label dataset**, by using the `label data` command, or by editing the *Label* field in the Data portion of the Properties window. When doing this in the main window, be sure that the Properties window is unlocked.

2. There is a variable label attached to each variable. Variable labels are how we would refer to the variable in normal, everyday conversation. Here they also contain information about the units of the variables. Variables can be labeled by selecting the variable in the Variables window and editing the *Label* field in the Properties window. You can also change a variable label by using the Variables Manager or by using the `label variable` command.

3. The `foreign` variable has an attached value label. Value labels allow numeric variables such as `foreign` to have words associated with numeric codes. The `describe` output tells you that the numeric variable `foreign` has value label `origin` associated with it. Although not revealed by `describe`, the variable `foreign` takes on the values 0 and 1, and the value label `origin` associates 0 with Domestic and 1 with Foreign. If you `browse` the data (see [GSW] **6 Using the Data Editor**), `foreign` appears to contain the values "Domestic" and "Foreign". The values in a variable are labeled in two stages. The value label must first be defined. This can be done in the Data Editor, or in the Variables Manager, or by selecting **Data > Data utilities > Label utilities > Manage value labels** or by typing the `label define` command. After the labels have been defined, they must be attached to the proper variables, either by selecting **Data > Data utilities > Label utilities > Assign value label to variables** or by using the `label values` command.

Note: It is not necessary for the value label to have a name different from that of the variable. You could just as easily have used a value label named `foreign`.

Labeling datasets and variables

We will now load the `afewcars.dta` dataset and give it proper labels. We will do this with the Command window to illustrate that it is simple to do in this fashion. Earlier in *Renaming and formatting variables* in [GSW] **6 Using the Data Editor**, we used the Data Editor to achieve a similar purpose. If you use the Data Editor for the material here, you will end up with the same commands in your log; we would like to illustrate a way to work directly with commands.

```
. use afewcars

. describe

Contains data from afewcars.dta
 Observations:              7
    Variables:              5                      21 Jan 2021 16:47
─────────────────────────────────────────────────────────────────────────
Variable       Storage    Display    Value
    name          type     format    label      Variable label
─────────────────────────────────────────────────────────────────────────
make            str18     %18s
price           float     %9.0g
mpg             float     %9.0g
weight          float     %9.0g
gear_ratio      float     %9.0g
─────────────────────────────────────────────────────────────────────────
Sorted by:

. label data "A few 1978 cars"

. label variable make "Make and model"

. label variable price "Price (USD)"

. label variable mpg "Mileage (miles per gallon)"

. label variable weight "Vehicle weight (lbs.)"

. label variable gear_ratio "Gear ratio"

. describe

Contains data from afewcars.dta
 Observations:              7                      A few 1978 cars
    Variables:              5                      21 Jan 2021 16:47
─────────────────────────────────────────────────────────────────────────
Variable       Storage    Display    Value
    name          type     format    label      Variable label
─────────────────────────────────────────────────────────────────────────
make            str18     %18s                   Make and model
price           float     %9.0g                  Price (USD)
mpg             float     %9.0g                  Mileage (miles per gallon)
weight          float     %9.0g                  Vehicle weight (lbs.)
gear_ratio      float     %9.0g                  Gear ratio
─────────────────────────────────────────────────────────────────────────
Sorted by:
     Note: Dataset has changed since last saved.

. save afewcars2
file afewcars2.dta saved
```

Labeling values of variables

We will now add a new indicator variable to the dataset that is 0 if the car was made in the United States and 1 if it was made in another country. Open the Data Editor and use your previously gained knowledge to add a `foreign` variable whose values match what is shown in this listing:

```
. list, separator(0)

            make    price    mpg   weight   gear_r~o   foreign

  1.    VW Rabbit    4697     25     1930       3.78         1
  2.      Olds 98    8814     21     4060       2.41         0
  3.   Chev. Monza   3667      .     2750       2.73         0
  4.                 4099     22     2930       3.58         0
  5.    Datsun 510   5079     24     2280       3.54         1
  6.   Buick Regal   5189     20     3280       2.93         0
  7.   Datsun 810    8129      .     2750       3.55         1
```

You can create this new variable in the Data Editor if you would like to work along. (See [GSW] **6 Using the Data Editor** for help with the Data Editor.) Though the definitions of the categories "0" and "1" are clear in this context, it still would be worthwhile to give the values explicit labels because it will make output clear to people who are not so familiar with antique automobiles. This is done with a value label.

We saw an example of creating and attaching a value label by using the point-and-click interface available in the Data Editor in *Changing data* in [GSW] **6 Using the Data Editor**. Here we will do it directly from the Command window.

```
. label define origin 0 "Domestic" 1 "Foreign"

. label values foreign origin

. describe
Contains data from afewcars2.dta
 Observations:              7                    A few 1978 cars
    Variables:              6                    21 Jan 2021 16:47
─────────────────────────────────────────────────────────────────────
Variable        Storage   Display    Value
    name           type    format    label      Variable label
─────────────────────────────────────────────────────────────────────
make            str18     %18s                  Make and model
price           float     %9.0g                 Price (USD)
mpg             float     %9.0g                 Mileage (miles per gallon)
weight          float     %9.0g                 Vehicle weight (lbs.)
gear_ratio      float     %9.0g                 Gear ratio
foreign         byte      %8.0g      origin
─────────────────────────────────────────────────────────────────────
Sorted by:
     Note: Dataset has changed since last saved.

. save afewcarslab
file afewcarslab.dta saved
```

From this example, we can see that a value label is defined via

> `label define` *labelname* *#* "*contents*" *#* "*contents*" ...

It can then be attached to a variable via

> `label values` *variablename labelname*

Once again, we need to save the dataset to be sure that we do not mistakenly lose the labels later. We saved this under a new filename because we have cleaned it up, and we would like to use it in the next chapter.

If you had wanted to define the value labels by using a point-and-click interface, you could do this with the Properties window in either the Main window or the Data Editor or by using the Variables Manager. See [GSW] **7 Using the Variables Manager** for more information.

There is more to value labels than what was covered here; see [U] **12.6.3 Value labels** for a complete treatment.

You may also add notes to your data and your variables. This feature was previously discussed in *Renaming and formatting variables* in [GSW] **6 Using the Data Editor** and *Managing notes* in [GSW] **7 Using the Variables Manager**. You can learn more about notes by typing `help notes`, or you can get the full story in [D] **notes**.

10   Listing data and basic command syntax

Command syntax

This chapter gives a basic lesson on Stata's command syntax while showing how to control the appearance of a data list.

As we have seen throughout this manual, you have a choice between using menus and dialogs and using the Command window. Although many find the menus more natural and the Command window baffling at first, some practice makes working with the Command window often much faster than using menus and dialogs. The Command window can become a faster way of working because of the clean and regular syntax of Stata commands. We will cover enough to get you started; help language has more information and examples, and [U] **11 Language syntax** has all the details.

The syntax for the list command can be seen by typing help list:

$$\underline{\text{l}}\text{ist}\ \big[\ \textit{varlist}\ \big]\ \big[\ \textit{if}\ \big]\ \big[\ \textit{in}\ \big]\ \big[\ ,\ \textit{options}\ \big]$$

Here is how to read this syntax:
- Anything inside square brackets is optional. For the list command,
    a. *varlist* is optional. A *varlist* is a list of variable names.
    b. *if* is optional. The if qualifier restricts the command to run only on those observations for which the qualifier is true. We saw examples of this in [GSW] **6 Using the Data Editor**.
    c. *in* is optional. The in qualifier restricts the command to run on particular observation numbers.
    d. , and *options* are optional. *options* are separated from the rest of the command by a comma.
- Optional pieces do not preclude one another unless explicitly stated. For the list command, it is possible to use a *varlist* with *if* and *in*.
- If a part of a word is underlined, the underlined part is the minimum abbreviation. Any abbreviation at least this long is acceptable.
    a. The l in list is underlined, so l, li, and lis are all equivalent to list.
- Anything not inside square brackets is required. For the list command, only the command itself is required.

Keeping these rules in mind, let's investigate how list behaves when called with different arguments. We will be using the dataset afewcarslab.dta from the end of the previous chapter.

list with a variable list

Variable lists (or *varlist*s) can be specified in a variety of ways, all designed to save typing and encourage good variable names.
- The *varlist* is optional for list. This means that if no variables are specified, it is equivalent to specifying all variables. Another way to think of it is that the default behavior of the command is to run on all variables unless restricted by a *varlist*.
- You can list a subset of variables explicitly, as in list make mpg price.
- There are also many shorthand notations:
    m* means all variables starting with m.
    price-weight means all variables from price through weight in the dataset order.
    ma?e means all variables starting with ma, followed by any character, and ending in e.

● You can list a variable by using an abbreviation unique to that variable, as in `list gear_r~o`. If the abbreviation is not unique, Stata returns an error message.

```
. list
```

	make	price	mpg	weight	gear_r~o	foreign
1.	VW Rabbit	4697	25	1930	3.78	Foreign
2.	Olds 98	8814	21	4060	2.41	Domestic
3.	Chev. Monza	3667	.	2750	2.73	Domestic
4.		4099	22	2930	3.58	Domestic
5.	Datsun 510	5079	24	2280	3.54	Foreign
6.	Buick Regal	5189	20	3280	2.93	Domestic
7.	Datsun 810	8129	.	2750	3.55	Foreign

```
. l make mpg price
```

	make	mpg	price
1.	VW Rabbit	25	4697
2.	Olds 98	21	8814
3.	Chev. Monza	.	3667
4.		22	4099
5.	Datsun 510	24	5079
6.	Buick Regal	20	5189
7.	Datsun 810	.	8129

```
. list m*
```

	make	mpg
1.	VW Rabbit	25
2.	Olds 98	21
3.	Chev. Monza	.
4.		22
5.	Datsun 510	24
6.	Buick Regal	20
7.	Datsun 810	.

```
. li price-weight
```

	price	mpg	weight
1.	4697	25	1930
2.	8814	21	4060
3.	3667	.	2750
4.	4099	22	2930
5.	5079	24	2280
6.	5189	20	3280
7.	8129	.	2750

```
. list ma?e

                make

  1.        VW Rabbit
  2.          Olds 98
  3.      Chev. Monza
  4.
  5.       Datsun 510

  6.       Buick Regal
  7.       Datsun 810

. l gear_r~o

            gear_r~o

  1.            3.78
  2.            2.41
  3.            2.73
  4.            3.58
  5.            3.54

  6.            2.93
  7.            3.55
```

list with if

The `if` qualifier uses a logical expression to determine which observations to use. If the expression is true, the observation is used in the command; otherwise, it is skipped. The operators whose results are either true or false are

<	less than
<=	less than or equal
==	equal
>	greater than
>=	greater than or equal
!=	not equal
&	and
|	or
!	not (logical negation)
()	parentheses are for grouping to specify order of evaluation

In the logical expressions, & is evaluated before | (similar to multiplication before addition in arithmetic). You can use this in your expressions, but it is often better to use parentheses to ensure that the expressions are evaluated in the proper order. See [U] **13.2 Operators** for complete details.

```
. list

                make    price    mpg    weight    gear_r~o    foreign

    1.      VW Rabbit     4697     25      1930        3.78    Foreign
    2.        Olds 98     8814     21      4060        2.41    Domestic
    3.    Chev. Monza     3667      .      2750        2.73    Domestic
    4.                    4099     22      2930        3.58    Domestic
    5.     Datsun 510     5079     24      2280        3.54    Foreign

    6.    Buick Regal     5189     20      3280        2.93    Domestic
    7.     Datsun 810     8129      .      2750        3.55    Foreign

. list if mpg > 22

                make    price    mpg    weight    gear_r~o    foreign

    1.      VW Rabbit     4697     25      1930        3.78    Foreign
    3.    Chev. Monza     3667      .      2750        2.73    Domestic
    5.     Datsun 510     5079     24      2280        3.54    Foreign
    7.     Datsun 810     8129      .      2750        3.55    Foreign

. list if (mpg > 22) & !missing(mpg)

                make    price    mpg    weight    gear_r~o    foreign

    1.      VW Rabbit     4697     25      1930        3.78    Foreign
    5.     Datsun 510     5079     24      2280        3.54    Foreign

. list make mpg price gear if (mpg > 22) | (price > 8000 & gear < 3.5)

                make     mpg    price    gear_r~o

    1.      VW Rabbit     25     4697        3.78
    2.        Olds 98     21     8814        2.41
    3.    Chev. Monza      .     3667        2.73
    5.     Datsun 510     24     5079        3.54
    7.     Datsun 810      .     8129        3.55

. list make mpg if mpg <= 22 in 2/4

            make    mpg

    2.    Olds 98     21
    4.                22
```

In the listings above, we see more examples of Stata treating missing numerical values as large values, as well as the care that should be taken when the if qualifier is applied to a variable with missing values. See [GSW] **6 Using the Data Editor**.

list with if, common mistakes

Here is a series of listings with common errors and their corrections. See if you can find the errors before reading the correct entry.

```
. list

             make    price    mpg    weight    gear_r~o    foreign

  1.     VW Rabbit     4697     25      1930        3.78     Foreign
  2.       Olds 98     8814     21      4060        2.41    Domestic
  3.   Chev. Monza     3667      .      2750        2.73    Domestic
  4.                   4099     22      2930        3.58    Domestic
  5.    Datsun 510     5079     24      2280        3.54     Foreign

  6.   Buick Regal     5189     20      3280        2.93    Domestic
  7.   Datsun 810     8129      .      2750        3.55     Foreign

. list if mpg=21
=exp not allowed
r(101);
```

The error arises because "equal" is expressed by ==, not by =. Corrected, it becomes

```
. list if mpg==21

          make    price    mpg    weight    gear_r~o    foreign

  2.    Olds 98     8814     21      4060        2.41    Domestic
```

Other common errors with logic:

```
. list if mpg==21 if weight > 4000
invalid syntax
r(198);
. list if mpg==21 and weight > 4000
invalid 'and'
r(198);
```

Joint tests are specified with &, not with the word and or multiple ifs. The if qualifier should be if mpg==21 & weight>4000, not if mpg==21 if weight>4000. Here is its correction:

```
. list if mpg==21 & weight > 4000

          make    price    mpg    weight    gear_r~o    foreign

  2.    Olds 98     8814     21      4060        2.41    Domestic
```

A problem with string variables:

```
. list if make==Datsun 510
Datsun not found
r(111);
```

Strings must be in double quotes, as in `make=="Datsun 510"`. Without the quotes, Stata thinks that `Datsun` is a variable that it cannot find. Here is the correction:

```
. list if make=="Datsun 510"

              make    price   mpg   weight   gear_r~o   foreign

  5.     Datsun 510    5079    24    2280       3.54    Foreign
```

Confusing value labels with strings:

```
. list if foreign=="Domestic"
type mismatch
r(109);
```

Value labels look like strings, but the underlying variable is numeric. Variable `foreign` takes on values 0 and 1 but has the value label that attaches 0 to "Domestic" and 1 to "Foreign" (see [GSW] **9 Labeling data**). To see the underlying numeric values of variables with labeled values, use the `label list` command (see [D] **label**), or investigate the variable with `codebook` *varname*. We can correct the error here by looking for observations where `foreign==0`.

There is a second construction that also allows the use of the value label directly.

```
. list if foreign==0

              make    price   mpg   weight   gear_r~o   foreign

  2.       Olds 98     8814    21    4060       2.41    Domestic
  3.    Chev. Monza    3667     .    2750       2.73    Domestic
  4.                   4099    22    2930       3.58    Domestic
  6.    Buick Regal    5189    20    3280       2.93    Domestic

. list if foreign=="Domestic":origin

              make    price   mpg   weight   gear_r~o   foreign

  2.       Olds 98     8814    21    4060       2.41    Domestic
  3.    Chev. Monza    3667     .    2750       2.73    Domestic
  4.                   4099    22    2930       3.58    Domestic
  6.    Buick Regal    5189    20    3280       2.93    Domestic
```

list with in

The `in` qualifier uses a *numlist* to give a range of observations that should be listed. *numlist*s have the form of one number or *first*/*last*. Positive numbers count from the beginning of the dataset. Negative numbers count from the end of the dataset. Here are some examples:

```
. list

             make    price    mpg    weight    gear_r~o    foreign

  1.     VW Rabbit     4697     25      1930        3.78     Foreign
  2.       Olds 98     8814     21      4060        2.41    Domestic
  3.   Chev. Monza     3667      .      2750        2.73    Domestic
  4.                   4099     22      2930        3.58    Domestic
  5.    Datsun 510     5079     24      2280        3.54     Foreign

  6.   Buick Regal     5189     20      3280        2.93    Domestic
  7.    Datsun 810     8129      .      2750        3.55     Foreign

. list in 1

            make    price    mpg    weight    gear_r~o    foreign

  1.   VW Rabbit     4697     25      1930        3.78    Foreign

. list in -1

             make    price    mpg    weight    gear_r~o    foreign

  7.   Datsun 810     8129      .      2750        3.55    Foreign

. list in 2/4

             make    price    mpg    weight    gear_r~o    foreign

  2.       Olds 98     8814     21      4060        2.41    Domestic
  3.   Chev. Monza     3667      .      2750        2.73    Domestic
  4.                   4099     22      2930        3.58    Domestic

. list in -3/-2

             make    price    mpg    weight    gear_r~o    foreign

  5.   Datsun 510     5079     24      2280        3.54     Foreign
  6.   Buick Regal     5189     20      3280        2.93    Domestic
```

Controlling the list output

The fine control over `list` output is exercised by specifying one or more options. You can use `sepby()` to separate observations by variable. `abbreviate()` specifies the minimum number of characters to abbreviate a variable name in the output. `divider` draws a vertical line between the variables in the list.

```
. sort foreign
. list ma p g f, sepby(foreign)

              make    price    gear_r~o     foreign

   1.                  4099        3.58    Domestic
   2.   Buick Regal    5189        2.93    Domestic
   3.   Chev. Monza    3667        2.73    Domestic
   4.        Olds 98    8814        2.41    Domestic

   5.   Datsun 510     5079        3.54     Foreign
   6.     VW Rabbit    4697        3.78     Foreign
   7.   Datsun 810     8129        3.55     Foreign

. list make weight gear, abbreviate(10)

              make    weight    gear_ratio

   1.                  2930         3.58
   2.   Buick Regal    3280         2.93
   3.   Chev. Monza    2750         2.73
   4.        Olds 98    4060         2.41
   5.   Datsun 510     2280         3.54

   6.     VW Rabbit    1930         3.78
   7.   Datsun 810     2750         3.55

. list, divider

              make    price   mpg   weight    gear_r~o     foreign

   1.                  4099    22     2930        3.58    Domestic
   2.   Buick Regal    5189    20     3280        2.93    Domestic
   3.   Chev. Monza    3667     .     2750        2.73    Domestic
   4.        Olds 98    8814    21     4060        2.41    Domestic
   5.   Datsun 510     5079    24     2280        3.54     Foreign

   6.     VW Rabbit    4697    25     1930        3.78     Foreign
   7.   Datsun 810     8129     .     2750        3.55     Foreign
```

The separator() option draws a horizontal line at specified intervals. When not specified, it defaults to a value of 5.

```
. list, separator(3)

               make    price    mpg   weight   gear_r~o    foreign

  1.                    4099     22     2930       3.58    Domestic
  2.    Buick Regal     5189     20     3280       2.93    Domestic
  3.    Chev. Monza     3667      .     2750       2.73    Domestic

  4.        Olds 98     8814     21     4060       2.41    Domestic
  5.     Datsun 510     5079     24     2280       3.54     Foreign
  6.      VW Rabbit     4697     25     1930       3.78     Foreign

  7.     Datsun 810     8129      .     2750       3.55     Foreign
```

Break

If you want to interrupt a Stata command, click on the **Break** button, ✖ .

It is always safe to click on the **Break** button. After you click on **Break**, the state of the system is the same as if you had never issued the original command.

11 Creating new variables

generate and replace

This chapter shows the basics of creating and modifying variables in Stata. We saw how to work with the Data Editor in [GSW] **6 Using the Data Editor**—this chapter shows how we would do this from the Command window. The two primary commands used for this are

- `generate` for creating new variables. It has a minimum abbreviation of `g`.
- `replace` for replacing the values of an existing variable. It may not be abbreviated because it alters existing data and hence can be considered dangerous.

The most basic form for creating new variables is `generate` *newvar* = *exp*, where *exp* is any kind of *expression*. Of course, both `generate` and `replace` can be used with `if` and `in` qualifiers. An expression is a formula made up of constants, existing variables, operators, and functions. Some examples of expressions (using variables from `auto.dta`) would be `2 + price`, `weight^2` or `sqrt(gear_ratio)`.

The operators defined in Stata are given in the table below:

	Arithmetic		Logical		Relational (numeric and string)
+	addition	!	not	>	greater than
−	subtraction	|	or	<	less than
*	multiplication	&	and	>=	> or equal
/	division			<=	< or equal
^	power			==	equal
				!=	not equal
+	string concatenation				

Stata has many mathematical, statistical, string, date, time-series, and programming functions. See `help functions` for the basics, and see the *Stata Functions Reference Manual* for a complete list and full details of all the built-in functions.

You can use menus and dialogs to create new variables and modify existing variables by selecting menu items from the **Data > Create or change data** menu. This feature can be handy for finding functions quickly. However, we will use the Command window for the examples in this chapter because we would like to illustrate simple usage and some pitfalls.

Stata has some utility commands for creating new variables:

- The `egen` command is useful for working across groups of variables or within groups of observations. See [D] **egen** for more information.
- The `encode` command turns categorical string variables into encoded numeric variables, while its counterpart `decode` reverses this operation. See [D] **encode** for more information.
- The `destring` command turns string variables that should be numeric, such as numbers with currency symbols, into numbers. To go from numbers to strings, the `tostring` command is useful. See [D] **destring** for more information.

We will focus our efforts on `generate` and `replace`.

generate

There are some details you should know about the generate command:

- The basic form of the generate command is generate *newvar* = *exp*, where *newvar* is a new variable name and *exp* is any valid expression. You will get an error message if you try to generate a variable that already exists.

- An algebraic calculation using a missing value yields a missing value, as does division by zero, the square root of a negative number, or any other computation which is impossible.

- If missing values are generated, the number of missing values in *newvar* is always reported. If Stata says nothing about missing values, then no missing values were generated.

- You can use generate to set the storage type of the new variable as it is generated. You might want to create an indicator (0/1) variable as a byte, for example, because it saves 3 bytes per observation over using the default storage type of float.

Below are some examples of creating new variables from the afewcarslab dataset, which we created in *Labeling values of variables* in [GSW] **9 Labeling data**. (To work along, start by opening the automobile dataset with sysuse auto. We are using a smaller dataset to make shorter listings.) The last example shows a way to generate an indicator variable for cars weighing more than 3,000 pounds. Logical expressions in Stata result in 1 for "true" and 0 for "false". The if qualifier is used to ensure that the computations are done only for observations where weight is not missing.

```
. use afewcarslab
(A few 1978 cars)

. list make mpg weight

            make    mpg   weight

  1.     VW Rabbit    25     1930
  2.       Olds 98    21     4060
  3.   Chev. Monza     .     2750
  4.                  22     2930
  5.    Datsun 510    24     2280

  6.   Buick Regal    20     3280
  7.    Datsun 810     .     2750


. * changing MPG to liters per 100km
. generate lphk = 3.7854 * (100 / 1.6093) / mpg
(2 missing values generated)

. label var lphk "Liters per 100km"

. * getting logarithms of price
. g lnprice = ln(price)

. * making an indicator of hugeness
. gen byte huge = weight >= 3000 if !missing(weight)

. l make mpg weight lphk lnprice huge

            make    mpg   weight       lphk    lnprice   huge

  1.     VW Rabbit    25     1930   9.408812   8.454679      0
  2.       Olds 98    21     4060   11.20097   9.084097      1
  3.   Chev. Monza     .     2750          .   8.207129      0
  4.                  22     2930   10.69183   8.318499      0
  5.    Datsun 510    24     2280   9.800845   8.532869      0

  6.   Buick Regal    20     3280   11.76101   8.554296      1
  7.    Datsun 810     .     2750          .   9.003193      0
```

replace

Whereas `generate` is used to create new variables, `replace` is the command used for existing variables. Stata uses two different commands to prevent you from accidentally modifying your data. The `replace` command cannot be abbreviated. Stata generally requires you to spell out completely any command that can alter your existing data.

```
. list make weight

              make    weight

    1.    VW Rabbit      1930
    2.      Olds 98      4060
    3.  Chev. Monza      2750
    4.                   2930
    5.   Datsun 510      2280

    6.  Buick Regal      3280
    7.   Datsun 810      2750

. * will give an error because weight already exists
. gen weight = weight/1000
variable weight already defined
r(110);

. * will replace weight in lbs by weight in 1000s of lbs
. replace weight = weight/1000
(7 real changes made)

. list make weight

              make    weight

    1.    VW Rabbit      1.93
    2.      Olds 98      4.06
    3.  Chev. Monza      2.75
    4.                   2.93
    5.   Datsun 510      2.28

    6.  Buick Regal      3.28
    7.   Datsun 810      2.75
```

Suppose that you want to create a new variable, `predprice`, which will be the predicted price of the cars in the following year. You estimate that domestic cars will increase in price by 5% and foreign cars, by 10%.

One way to create the variable would be to first use `generate` to compute the predicted domestic car prices. Then use `replace` to change the missing values for the foreign cars to their proper values.

```
. gen predprice = 1.05*price if foreign==0
(3 missing values generated)

. replace predprice = 1.10*price if foreign==1
(3 real changes made)

. list make foreign price predprice, nolabel

             make     foreign     price     predpr~e

  1.    VW Rabbit          1       4697       5166.7
  2.       Olds 98         0       8814       9254.7
  3.    Chev. Monza        0       3667      3850.35
  4.                       0       4099      4303.95
  5.    Datsun 510         1       5079       5586.9

  6.    Buick Regal        0       5189      5448.45
  7.    Datsun 810         1       8129       8941.9
```

Of course, because foreign is an indicator variable, we could generate the predicted variable with one command:

```
. gen predprice2 = (1.05 + 0.05*foreign)*price

. list make foreign price predprice predprice2, nolabel

             make     foreign     price     predpr~e     predpr~2

  1.    VW Rabbit          1       4697       5166.7       5166.7
  2.       Olds 98         0       8814       9254.7       9254.7
  3.    Chev. Monza        0       3667      3850.35      3850.35
  4.                       0       4099      4303.95      4303.95
  5.    Datsun 510         1       5079       5586.9       5586.9

  6.    Buick Regal        0       5189      5448.45      5448.45
  7.    Datsun 810         1       8129       8941.9       8941.9
```

generate with string variables

Stata is smart. When you generate a variable and the expression evaluates to a string, Stata creates a string variable with a storage type as long as necessary, and no longer than that. `where` is a `str1` in the following example:

```
. list make foreign

              make      foreign

  1.     VW Rabbit      Foreign
  2.        Olds 98     Domestic
  3.    Chev. Monza     Domestic
  4.                    Domestic
  5.     Datsun 510      Foreign

  6.    Buick Regal     Domestic
  7.     Datsun 810      Foreign

. gen where = "D" if foreign=="Domestic":origin
(3 missing values generated)
. replace where = "F" if foreign=="Foreign":origin
(3 real changes made)
. list make foreign where

              make      foreign    where

  1.     VW Rabbit      Foreign       F
  2.        Olds 98     Domestic      D
  3.    Chev. Monza     Domestic      D
  4.                    Domestic      D
  5.     Datsun 510      Foreign       F

  6.    Buick Regal     Domestic      D
  7.     Datsun 810      Foreign       F

. describe where
Variable      Storage   Display    Value
    name         type    format    label       Variable label
───────────────────────────────────────────────────────────────────
where             str1    %9s
```

Stata has some useful tools for working with string variables. Here we split the `make` variable into make and model and then create a variable that has the model together with where the model was manufactured:

```
. gen model = usubstr(make, ustrpos(make," ")+1,.)
(1 missing value generated)

. gen modelwhere = model + " " + where

. list make where model modelwhere

             make    where    model   modelw~e

   1.    VW Rabbit        F   Rabbit   Rabbit F
   2.      Olds 98        D       98       98 D
   3.   Chev. Monza       D    Monza    Monza D
   4.                     D                   D
   5.    Datsun 510       F      510      510 F

   6.   Buick Regal       D    Regal    Regal D
   7.    Datsun 810       F      810      810 F
```

There are a few things to note about how these commands work:

1. `ustrpos(`s_1`,`s_2`)` produces an integer equal to the first character in the string s_1 at which the string s_2 is found or 0 if it is not found. In this example, `ustrpos(make," ")` finds the position of the first space in each observation of `make`.

2. `usubstr(`s`,`$start$`,`len`)` produces a string of length len characters, beginning at character $start$ of string s. If $c_1 = .$, the result is the string from character $start$ to the end of string s.

3. Putting 1 and 2 together: `usubstr(`s`,ustrpos(`s`," ")+1,.)` will always give the string s with its first word removed. Because `make` contains both the make and the model of each car, and `make` never contains a space in this dataset, we have found each car's model.

4. The operator "`+`", when applied to string variables, will concatenate the strings (that is, join them together). The expression `"this" + "that"` results in the string `"thisthat"`. When the variable `modelwhere` was generated, a space (`" "`) was added between the two strings.

5. The missing value for a string is nothing special—it is simply the empty string `""`. Thus the value of `modelwhere` for the car with no make or model is `" D"` (note the leading space).

6. If your strings might contain Unicode characters, use the Unicode versions of the string functions, as shown above. See [U] **12.4.2 Handling Unicode strings**.

12 Deleting variables and observations

clear, drop, and keep

In this chapter, we will present the tools for paring observations and variables from a dataset. We saw how to do this using the Data Editor in [GSW] **6 Using the Data Editor**; this chapter presents the methods for doing so from the Command window.

There are three main commands for removing data and other Stata objects, such as value labels, from memory: `clear`, `drop`, and `keep`. Remember that they affect only what is in memory. None of these commands alter anything that has been saved to disk.

clear and drop _all

Suppose that you are working on an analysis or a simulation and that you need to clear out Stata's memory so that you can impute different values or simulate a new dataset. You are not interested in saving any of the changes you have made to the dataset in memory—you would just like to have an empty dataset. What you do depends on how much you want to clear out: at any time, you can have not only data but also metadata such as value labels, stored results from previous commands, and stored matrices. The `clear` command will let you carefully clear out data or other objects; we are interested only in simple usage here. For more information, see `help clear` and [D] **clear**.

If you type the command `clear` into the Command window, it will remove all variables and value labels. In basic usage, this is typically enough. It has the nice property that it does not remove any stored results, so you can load a new dataset and predict values by using stored estimation results from a model fit on a previous dataset. See `help postest` and [U] **20 Estimation and postestimation commands** for more information.

If you want to be sure that everything is cleared out, use the command `clear all`. This command will clear Stata's memory of data and all auxiliary objects so that you can start with a clean slate. The first time you use `clear all` while you have a graph or dialog open, you may be surprised when that graph or dialog closes; this is necessary so that Stata can free all memory being used.

If you want to get rid of just the data and nothing else, you can use the command `drop _all`.

drop

The `drop` command is used to remove variables or observations from the dataset in memory.
- If you want to drop variables, use `drop` *varlist*.
- If you want to drop observations, use `drop` with an `if` or an `in` qualifier or both.

We will use the `afewcarslab` dataset to illustrate `drop`:

```
. use afewcarslab
(A few 1978 cars)
. list
```

	make	price	mpg	weight	gear_r~o	foreign
1.	VW Rabbit	4697	25	1930	3.78	Foreign
2.	Olds 98	8814	21	4060	2.41	Domestic
3.	Chev. Monza	3667	.	2750	2.73	Domestic
4.		4099	22	2930	3.58	Domestic
5.	Datsun 510	5079	24	2280	3.54	Foreign
6.	Buick Regal	5189	20	3280	2.93	Domestic
7.	Datsun 810	8129	.	2750	3.55	Foreign

```
. drop in 1/3
(3 observations deleted)
. list
```

	make	price	mpg	weight	gear_r~o	foreign
1.		4099	22	2930	3.58	Domestic
2.	Datsun 510	5079	24	2280	3.54	Foreign
3.	Buick Regal	5189	20	3280	2.93	Domestic
4.	Datsun 810	8129	.	2750	3.55	Foreign

```
. drop if mpg > 21
(3 observations deleted)
. list
```

	make	price	mpg	weight	gear_r~o	foreign
1.	Buick Regal	5189	20	3280	2.93	Domestic

```
. drop gear_ratio
. list
```

	make	price	mpg	weight	foreign
1.	Buick Regal	5189	20	3280	Domestic

```
. drop m*
. list
```

	price	weight	foreign
1.	5189	3280	Domestic

These changes are only to the data in memory. If you want to make the changes permanent, you need to save the dataset.

keep

keep tells Stata to drop all variables except those specified explicitly or through the use of an `if` or `in` expression. Just like `drop`, `keep` can be used with *varlist* or with qualifiers but not with both at once. We use a `clear` command at the start of this example so that we can reload the `afewcarslab` dataset:

```
. clear
. use afewcarslab
(A few 1978 cars)
. list
```

	make	price	mpg	weight	gear_r~o	foreign
1.	VW Rabbit	4697	25	1930	3.78	Foreign
2.	Olds 98	8814	21	4060	2.41	Domestic
3.	Chev. Monza	3667	.	2750	2.73	Domestic
4.		4099	22	2930	3.58	Domestic
5.	Datsun 510	5079	24	2280	3.54	Foreign
6.	Buick Regal	5189	20	3280	2.93	Domestic
7.	Datsun 810	8129	.	2750	3.55	Foreign

```
. keep in 4/7
(3 observations deleted)
. list
```

	make	price	mpg	weight	gear_r~o	foreign
1.		4099	22	2930	3.58	Domestic
2.	Datsun 510	5079	24	2280	3.54	Foreign
3.	Buick Regal	5189	20	3280	2.93	Domestic
4.	Datsun 810	8129	.	2750	3.55	Foreign

```
. keep if mpg <= 21
(3 observations deleted)
. list
```

	make	price	mpg	weight	gear_r~o	foreign
1.	Buick Regal	5189	20	3280	2.93	Domestic

```
. keep m*
. list
```

	make	mpg
1.	Buick Regal	20

13 Using the Do-file Editor—automating Stata

The Do-file Editor

Stata comes with an integrated text editor called the Do-file Editor, which can be used for many tasks. It gets its name from the term *do-file*, which is a file containing a list of commands for Stata to run (called a batch file or a script in other settings). See [U] **16 Do-files** for more information. Although the Do-file Editor has advanced features that can help in writing such files, it can also be used to build up a series of commands that can then be submitted to Stata all at once. This feature can be handy when writing a loop to process multiple variables in a similar fashion or when doing complex, repetitive tasks interactively.

To get the most from this chapter, you should work through it at your computer. Start by opening the Do-file Editor, either by clicking on the **Do-file Editor** button, 📝 , or by typing `doedit` in the Command window and pressing *Enter*.

The Do-file Editor toolbar

The Do-file Editor has 15 buttons. Many of the buttons share a similar purpose with their look-alikes in the main Stata toolbar.

If you ever forget what a button does, hover the mouse pointer over a button, and a tooltip will appear.

      New: Open a new do-file in a new tab in the Do-file Editor.

      Open: Open a do-file from disk in a new tab in the Do-file Editor.

      Save: Save the current file to disk.

      Print: Print the contents of the Do-file Editor.

      Find: Open the *Find* dialog for finding text.

      Cut: Cut the selected text and put it in the Clipboard.

      Copy: Copy the selected text to the Clipboard.

      Paste: Paste the text from the Clipboard into the current document.

      Undo: Undo the last change.

↪   **Redo**: Undo the last undo.

🔖±   **Toggle bookmark**: Turn on or off the bookmark on the current line. Bookmarks are a way to move quickly within the do-file. They are quite useful in long do-files or when debugging.

🔖↑   **Previous bookmark**: Go to the previous bookmark (if any).

🔖↓   **Next bookmark**: Go to the next bookmark (if any).

⦿   **Show file in Viewer**: Show the contents of the do-file in a Viewer window. This is worthwhile when editing files that contain SMCL tags, such as log files or help files.

▶   **Execute (do)**: Run the commands in the do-file, showing all commands and their output. If text is highlighted, the button becomes the **Execute selection (do)** button and will run only the selected lines, showing all output. We will refer to this as the **Do** button.

Using the Do-file Editor

Suppose that we would like to analyze fuel usage for 1978 automobiles in a manner similar to what we did in [GSW] **1 Introducing Stata—sample session**. We know that we will be issuing many commands to Stata during our analysis and that we want to be able to reproduce our work later without having to type each command again.

We can do this easily in Stata: simply save a text file containing the commands. When that is done, we can tell Stata to run the file and execute each command in sequence. Such a file is known as a Stata *do-file*; see [U] **16 Do-files**.

To analyze fuel usage of 1978 automobiles, we would like to create a new variable containing gallons per mile. We would like to see how that variable changes in relation to vehicle weight for both domestic and imported cars. Performing a regression with our new variable would be a good first step.

To get started, click on the **Do-file Editor** button to open the Do-file Editor. After the Do-file Editor opens, type the commands below into the Do-file Editor. Purposely misspell the name of the `foreign` variable on the fifth line. (We are intentionally making some common mistakes and then pointing you to the solutions. This will save you time later.)

```
* an example do-file
sysuse auto
generate gp100m = 100/mpg
label var gp100m "Gallons per 100 miles"
regress gp100m weight foreing
```

Here is what your Do-file Editor should look like now:

```
Do-file Editor - Untitled*                                    —    □    ✕
File   Edit   View   Language   Project   Tools

  Untitled*   ✕

1        * an example do-file
2        sysuse auto
3        generate gp100m = 100/mpg
4        label var gp100m "Gallons per 100 miles"
5        regress gp100m weight foreing
6

                                                   Line: 6, Col: 1   CAP   NUM   OVR
```
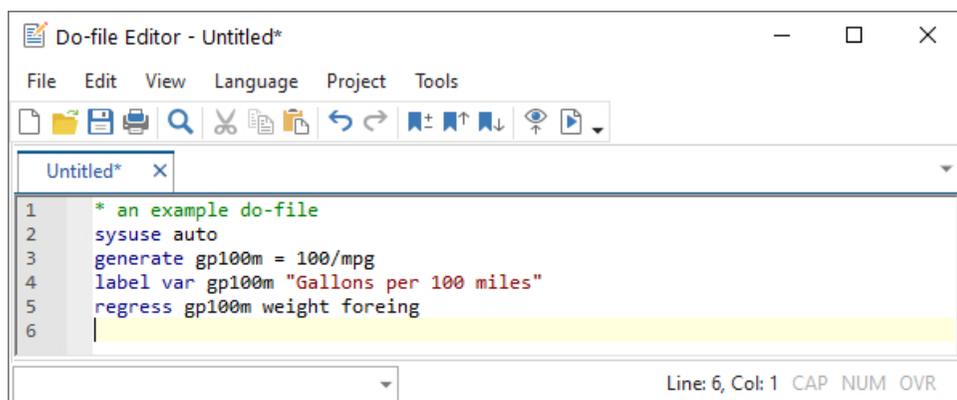
You will notice that the color of the text changes as you type. The different colors are examples of the Do-file Editor's syntax highlighting. The colors and text properties of the syntax elements can be changed by selecting **Edit > Preferences...** from the Do-file Editor menu bar and then clicking on the **Colors** tab in the resulting window.

Syntax highlighting extends beyond highlighting Stata commands. You can switch the syntax highlighting from Stata by going to the **Language** menu and choosing the language you would like. The **Language** menu includes a selection for Markdown because Stata can process Markdown to create dynamic documents. See [RPT] **dyndoc** for more information. This menu also contains selections for Python and Java because Stata has both Python integration and Java integration. See [P] **PyStata integration** and [P] **Java integration** for more information. Stata will default to the proper language based on the extension of the file you are editing, but if the file has not been saved yet, you will need to tell it what language to choose.

Also note that if you pause briefly as you type, the Do-file Editor will allow autocompletion of words that are already in the do-file. Once the suggestions appear, more typing will narrow down the possibilities. You can navigate the suggestions using the up- and down-arrow keys or keep typing to narrow them to a single word. Once you have the word you like, pressing *Enter* will place the word in your do-file.

Click on the **Do** button, ▶ , to execute the commands. Stata executes the commands in sequence, and the results appear in the Results window:

```
. do C:\Users\Stata\AppData\Local\Temp\STD08000000.tmp

. * an example do-file
. sysuse auto
(1978 automobile data)

. generate gp100m = 100/mpg

. label var gp100m "Gallons per 100 miles"

. regress gp100m weight foreing
variable foreing not found
r(111);

.
end of do-file
```

The do `"C:\ ... "` command is how Stata executes the commands in the Do-file Editor. Stata saves the commands from a do-file with unsaved changes to a temporary file and issues the `do` command to execute them. Everything worked as planned until Stata saw the misspelled variable. The first three commands were executed, but an error was produced on the fourth. Stata does not know of a variable named `foreing`. We need to go back to the Do-file Editor and change the misspelled variable name to `foreign` in the last line:

```
Do-file Editor - Untitled*                              —   □   ✕

File   Edit   View   Language   Project   Tools

1    * an example do-file
2    sysuse auto
3    generate gp100m = 100/mpg
4    label var gp100m "Gallons per 100 miles"
5    regress gp100m weight foreign
6    |
7
                                                    Line: 6, Col: 1  CAP  NUM  OVR
```

Click on the **Do** button again. Alas, Stata now fails on the first line—it will not overwrite the dataset in memory that we changed.

```
. do C:\Users\Stata\AppData\Local\Temp\STD08000000.tmp

. * an example do-file
. sysuse auto
no; dataset in memory has changed since last saved
r(4);

.
end of do-file
```

We now have a choice for what we should do:

- We can put a `clear` command in our do-file as the very first command. This automatically clears out Stata's memory before the do-file tries to load `auto.dta`. This is convenient but dangerous because it defeats Stata's protection against throwing away changes without warning.
- We can type a `clear` command in the Command window to manually clear the dataset and then process the do-file again. This process can be aggravating when building a complicated do-file.

Here is some advice: Automatically clear Stata's memory while debugging the do-file. Once the do-file is in its final form, decide the context in which it will be used. If it will be used in a highly automated environment (such as when certifying), the do-file should still automatically clear Stata's memory. If it will be used rarely, do not clear Stata's memory. This decision will save much heartache. We will add a `clear` option to the `sysuse` command to automatically clear the dataset in Stata's memory before the do-file runs:

```
Do-file Editor - Untitled*                                      —   □   ✕
File   Edit   View   Language   Project   Tools
```

```
Untitled*   ✕
1       * an example do-file
2       sysuse auto, clear
3       generate gp100m = 100/mpg
4       label var gp100m "Gallons per 100 miles"
5       regress gp100m weight foreign
6
7
                                              Line: 6, Col: 1  CAP  NUM  OVR
```

The do-file now runs well, as clicking on the **Do** button shows:

```
. do C:\Users\Stata\AppData\Local\Temp\STD08000000.tmp

. * an example do-file
. sysuse auto, clear
(1978 automobile data)

. generate gp100m = 100/mpg

. label var gp100m "Gallons per 100 miles"

. regress gp100m weight foreign

      Source |       SS       df       MS              Number of obs =      74
-------------+------------------------------           F(  2,    71) =  113.97
       Model | 91.1761694        2  45.5880847         Prob > F      =  0.0000
    Residual | 28.4000913       71   .400001287        R-squared     =  0.7625
-------------+------------------------------           Adj R-squared =  0.7558
       Total | 119.576261       73  1.63803097         Root MSE      =  .63246

-------------------------------------------------------------------------------
       p100m | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+-----------------------------------------------------------------
      weight |    .0016254   .0001183    13.74   0.000     .0013896    .0018612
     foreign |    .6220535   .1997381     3.11   0.003     .2237871    1.02032
       _cons |   -.0734839   .4019932    -0.18   0.855    -.8750354    .7280677
-------------------------------------------------------------------------------

.
end of do-file
```

You might want to select **File > Save as...** to save this do-file from the Do-file Editor. Later, you could select **File > Open** to open it and then add more commands as you move forward with your analysis. By saving the commands of your analysis in a do-file as you go, you do not have to worry about retyping them with each new Stata session. Think hard about removing the `clear` option from the first command.

After you have saved your do-file, you can execute the commands it contains by typing do *filename*, where the *filename* is the name of your do-file.

The File menu

The **File** menu of the Do-file Editor includes standard features found in most text editors. You may choose any of these menu items: create a **New > Do-file**, **Open** an existing file, **Save** the current file, save the current file under a new name with **Save as...**, or **Print** the current file. There are also buttons on the Do-file Editor's toolbar that correspond to these features.

Finally, you can create a **New > Project...** to keep track of collections of files used in a project. These can be do-files, data files, graph files, or any other files you like. For more information on the Project Manager, see [P] **Project Manager**.

The Edit menu

The **Edit** menu of the Do-file Editor includes the standard **Undo**, **Redo**, **Cut**, **Copy**, **Paste**, **Delete**, and **Find** capabilities. There are also buttons on the Do-file Editor's toolbar for easy access to these capabilities. There are several other **Edit** menu features that you might find useful:

- You can select **Insert file...** to insert the contents of another file at the current cursor position in the Do-file Editor.
- You can select the current line with **Select line**.
- You can delete the current line with **Delete line**.
- **Find > Go to line...** will allow you to jump to a specific line number. The line numbers are displayed at the left and the lower-right of the Do-file Editor window.
- **Advanced** leads to a submenu with some programmer's friends:
  - **Shift right** indents the selection by one tab.
  - **Shift left** unindents the selection by one tab.
  - **Re-indent** indents the selection according to its nesting within blocks and programs.
  - **Toggle comment** toggles //-style comments at the start of the selected lines.
  - **Add block comment** puts a /* before and a */ after the selected region, commenting it out.
  - **Remove block comment** undoes the above.
  - **Make selection uppercase** converts the selection to all capital letters.
  - **Make selection lowercase** converts the selection to all lowercase letters.
  - **Complete word** attempts to complete the current word based on words that are already in the do-file. If there are multiple possibilities, all will be shown. You can either pick the completion you would like or keep typing to narrow the choices.
  - **Convert to UTF-8...** converts the current file to UTF-8 encoding.
  - **Convert line endings to macOS/Unix format (\n)** converts the line endings for the current file to macOS/Unix format.
  - **Convert line endings to Windows format (\r\n)** converts the line endings for the current file to Windows format.
  - **Convert tabs to spaces** replaces any tab characters with spaces, leaving the spacing as it currently appears.
  - **Convert leading spaces to tabs** converts any spaces at the start of lines to tab characters. The number of spaces per tab is determined by a preference setting.
  - **Convert all spaces to tabs** converts spaces to tab characters wherever possible. The number of spaces per tab is determined by a preference setting.

Matching and balancing of parentheses ( ), braces { }, and brackets [ ] are also available from the **Edit** menu. When you select **Edit > Find > Match brace**, the Do-file Editor looks at the character immediately to the left and right of the cursor. If either is one of the characters that the editor can match, the editor will find the matching character and place the cursor immediately in front of it. If there is no match, the cursor will not move.

When you select **Edit > Find > Balance braces**, the Do-file Editor looks to the left and right of the current cursor position or selection and creates a selection that includes the narrowest level of matching braces. If you select **Balance braces** again, the editor will expand the selection to include the next level of matching braces. If there is no match, the cursor will not move. Balancing braces is useful for working with blocks of code defined by loops or if commands. See [P] **foreach**, [P] **forvalues**, [P] **while**, and [P] **if** for more information.

Balance braces is easier to explain with an example. Type {now {is the} time} in the Do-file Editor. Place the cursor between the words is and the. Select **Edit > Find > Balance braces**. The Do-file Editor will select {is the}. If you select **Balance braces** again, the Do-file Editor will select {now {is the} time}.

Text in Stata strings can include Unicode characters and is encoded as UTF-8 (see [U] **12.4.2 Handling Unicode strings**). However, you may have do-files, ado-files, or other text files that you used with Stata 13 or earlier, and those files contain characters other than plain ASCII such as accented characters, Chinese, Japanese, or Korean (CJK) characters, Cyrillic characters, and the like. If you open a file that is not encoded in UTF-8, Stata prompts you to specify the encoding for the file so that it can convert the file to UTF-8. If you cancel the conversion or choose the wrong encoding, you can try the conversion again later using **Convert to UTF-8**. The conversion to UTF-8 can be undone by using **Edit > Undo** and is not permanent until you save the do-file. For Stata datasets with characters not encoded in UTF-8 or for bulk conversion of multiple Stata files, you should use the unicode translate command.

Editing tip: You can click on the left margin near a line number to select the entire line and the end-of-line characters. Doing so makes it easy to delete lines or cut lines and paste them elsewhere. You can click and drag within the line-number column to select a range of complete lines.

The View menu

The View menu of the Do-File Editor allows you to zoom in and out or display special characters such as tabs and line endings.

The Tools menu

You have already learned about the **Do** button. Selecting **Tools > Execute (do)** is equivalent to clicking on the **Execute (do)** button.

Selecting **Tools > Execute (do) from top** will send all the commands from the first line to the current line to the Command window. This method is a quick way to run a part of a do-file.

Selecting **Tools > Execute (do) to bottom** will send all the commands from the current line through the end of the contents of the Do-file Editor to the Command window. This method is a quick way to run a part of a do-file.

Selecting **Tools > Execute quietly (run)** is equivalent to **Tools > Execute (do)** but the commands will be executed quietly; that is, no output will be displayed in the Command window.

Selecting **Tools > Execute (include)** is similar to clicking on the **Execute (do)** button with one major difference: local macros defined in the current session can be expanded in the commands being executed.

Do is equivalent to Stata's do command, whereas **Execute (include)** is equivalent to Stata's include command. See [U] **16 Do-files** for a complete discussion.

You can also preview files in the Viewer by selecting **Tools > Show file in Viewer** or by clicking on the **Show file in Viewer** button, 👁 . This feature is useful when working with files that use Stata's SMCL tags, such as when writing help files or editing log files.

Saving interactive commands from Stata as a do-file

While working interactively with Stata, you might decide that you would like to rerun the last several commands that you typed interactively. From the History window, you can send highlighted commands or even the entire contents to the Do-file Editor. You can also save commands as a do-file and open that file in the Do-file Editor. You can copy a command from a dialog (rather than submit it) and paste it into the Do-file Editor. See [GSW] **6 Using the Data Editor** for details. Also see [R] **log** for information on the cmdlog command, which allows you to log all commands that you type in Stata to a do-file.

Navigating your do-file

When working with long files, bookmarks allow you to easily navigate through your do-file. By placing a bookmark before important sections in your do-file, you can return to those sections more easily later. You can add a bookmark line by using **Edit > Toggle bookmark**, clicking the **Toggle bookmark** button on the toolbar, or by manually typing a line beginning with a special comment, **#. All other text on the rest of the line is treated as the title of the bookmark. You cannot have ado code on the same line as the bookmark comment, or the bookmark comment will be ignored.

You can move between bookmarks using the options in the **Edit** menu, the equivalent buttons on the Do-file Editor toolbar, or the Navigation Control. The Navigation Control of the Do-file Editor allows you to move between bookmarks as well as programs that you have defined in your do-file. When you select a program or a bookmark from the Navigation Control, you will jump directly to the position of that program or bookmark in your do-file.
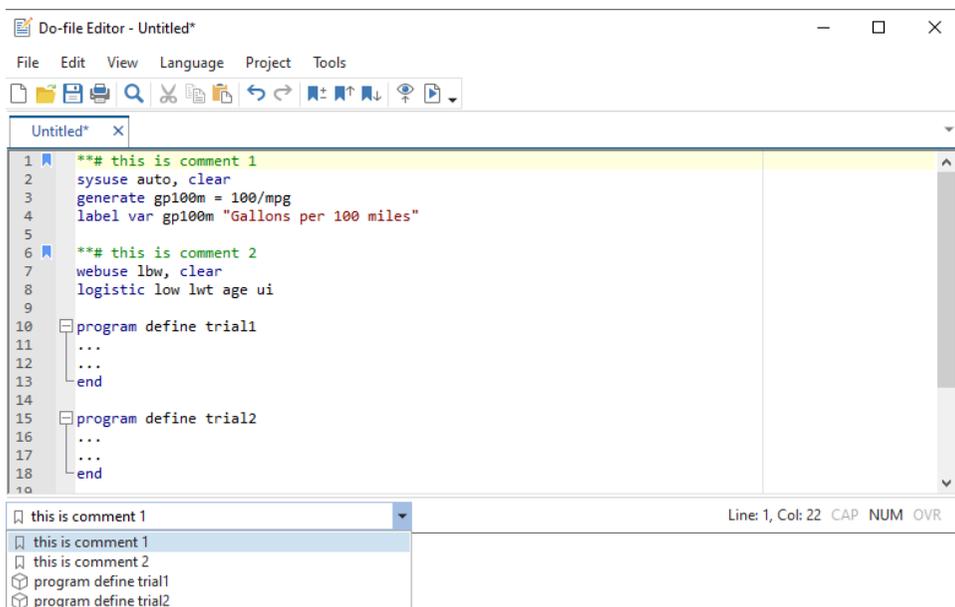
Bookmarks can also be removed with the **Toggle bookmark** option in the **Edit** menu or toolbar as well as simply deleting the line. You can also add and remove bookmarks by clicking in the bookmark margin next to the line you want to add the bookmark before. When you add a bookmark line, the bookmark icon will be added in the bookmark margin to make it more visibly obvious for you while scrolling.

Projects

For advanced users managing many files as part of a project, Stata has a Project Manager that uses the Do-file Editor. For more information on the Project Manager, see [P] **Project Manager**.

14   Graphing data

Working with graphs

Stata has a rich system for graphical representation of data. The main command for creating graphs is unsurprisingly named `graph`. Behind this plain name is a wealth of tools. In this chapter, we will make one simple graph to point out the basics of the Graph window. See the [G] *Stata Graphics Reference Manual* for more information about all aspects of working with graphs.

A simple graph example

In the sample session of [GSW] **1 Introducing Stata—sample session**, we made a scatterplot, added a fitted regression line, and made a grid of scatterplots to allow comparisons across groups. Here, using the automobile dataset, we make a simple box plot that shows the displacements of the cars' engines and how they compare across repair records within the place of manufacture of the cars. Start by loading the dataset by typing `sysuse auto` in the Command window and pressing *Enter*.

We select **Graphics > Box plot**, choose or type `displacement` in the *Variables* field on the **Main** tab, click on the **Categories** tab, check the *Group 1* checkbox and enter `rep78` for the first grouping variable, and check the *Group 2* checkbox and enter `foreign` for the second grouping variable. Finally, we click on the **Submit** button so that we could easily make changes to the graph if need be. After we look at the graph, we realize that we forgot the title. We close the Graph window, click on the **Titles** tab of the *graph box* dialog, type the title `Displacement across repairs within origin`, and click on the **Submit** button again.

The Graph window comes up, showing us our nicely titled graph:
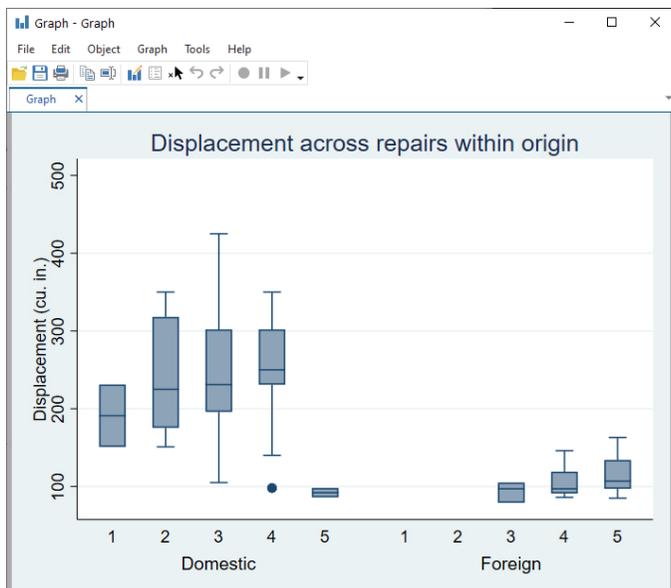
Graph window

When the Graph window comes up, it shows our graph in a window with a toolbar. The first four buttons are familiar to us from other Stata windows: **Open**, **Save**, **Print**, and **Copy**. The next two buttons are new:

 Rename graph: This button allows the graph to be renamed. Why would you do this? If you would like to have multiple graphs open at once, the graphs need to be named. So you can click on the **Rename graph** button to give a graph a name. This graph will then remain open when you create your next graph.

 Graph Editor: Stata has a Graph Editor that allows you to manipulate and edit your graph. This feature will be introduced in the next chapter.

The inactive buttons to the right of the **Graph Editor** button are used by the Graph Editor, so their meanings will become clear in the next chapter.

We decide that we like this graph and would like to save it. We can save it either by clicking on the **Save** button and choosing a name and a location or by right-clicking on the Graph window itself and selecting **Save as...**.

Saving and printing graphs

You can save a graph once it is displayed by right-clicking on its window and selecting **Save as...**. You can print a graph by right-clicking on its window and selecting **Print...**. You can also use the **File** menu to save or print a graph. We recommend that you always right-click on a graph to save or print it to ensure that the correct graph is selected.

Right-clicking on the Graph window

Right-clicking on the Graph window displays a menu from which you can select the following:

- **Save as...** to save the graph to disk.
- **Copy** to copy the graph to the Clipboard.
- **Start Graph Editor** to start the Graph Editor.
- **Preferences...** to edit the preferences for graphs.
- **Print...** to print the graph.

The Graph button

The **Graph** button, , is located on the main window's toolbar. The button has two parts, an icon and an arrow. Clicking on the icon brings the topmost Graph window to the front of all other windows. Clicking on the arrow displays a menu of open graphs. Selecting a graph from the menu brings that graph to the front of all other windows. If you close the Graph window, you can reopen it only by reissuing a Stata command that draws a new graph.

15   Editing graphs

The Graph Editor

With Stata's Graph Editor, you can change almost anything on your graph; you can add text, lines, arrows, and markers wherever you like.

We will first make a graph to edit and will then point out the tools in the Graph Editor. Start by opening the automobile dataset: `sysuse auto`. Here is the command that we will use to make the graph:

```
. scatter mpg weight, name(mygraph) title(Mileage vs. vehicle weight)
```

Start the Editor by right-clicking on your graph and selecting **Start Graph Editor**. Click once on the title of the graph. Here is a picture of the Graph Editor with its elements labeled.
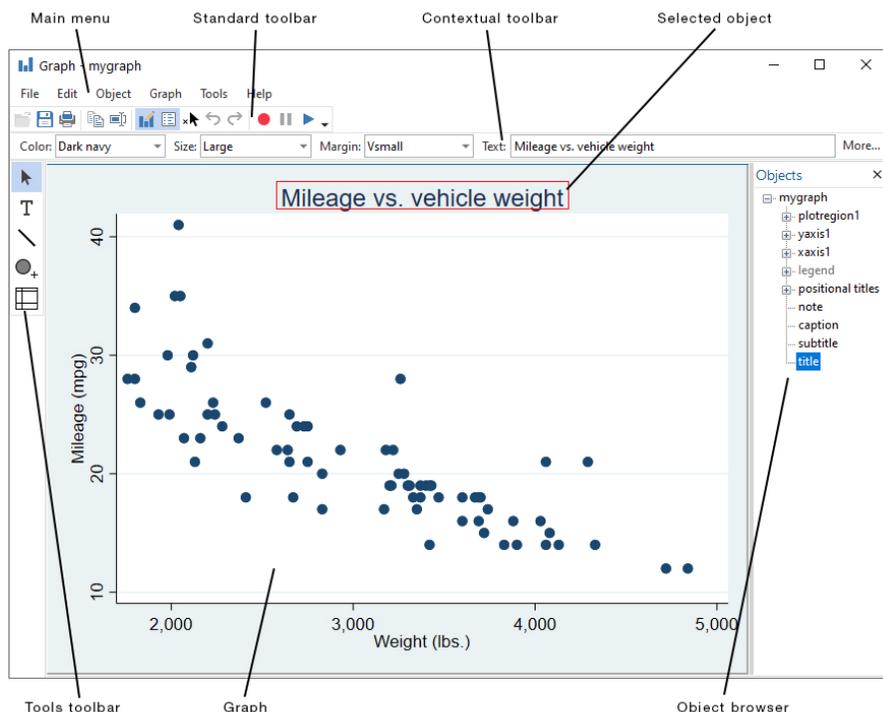
Select any of the tools along the left of the Graph Editor window to edit the graph. The Pointer (Select tool), ⬆, is selected by default.

You can change the properties of objects or drag them to new locations by using the Pointer. As you select objects with the Pointer, a Contextual Toolbar will appear just above the graph. In the above example, the title of the graph is selected, so the Contextual Toolbar has controls that are relevant for editing titles. You can use any of the controls on the Contextual Toolbar to immediately change the most important properties of the selected object. Right-click on an object to access more properties and operations. Hold the *Shift* key when dragging objects to constrain the movement to horizontal or vertical directions.

Add text, lines, or markers (with optional labels) to your graph by using the three *Add...* tools— T , ╲ , and ⬤₊ . Lines can be changed to arrows by using the Contextual Toolbar. If you do not like the default properties, simply change their settings in the Contextual Toolbar before adding the text, line, or marker. The new settings will then be applied to all added objects, even in future Stata sessions.

Do not be afraid to try things. If you do not like a result, change it back by using the same tool or by clicking on the **Undo** button, ↶ , in the Standard Toolbar for the Graph Editor (below the main menu). **Edit > Undo** in the main menu does the same thing.

Remember to reselect the Pointer tool when you want to drag objects or change their properties.

You can move objects on the graph and have the rest of the objects adjust their position to accommodate the move with the Grid edit tool, ▦ . With this tool, you are repositioning objects in the underlying grid that holds the objects in the graph. Some graphs, for example, by graphs, are composed of nested grids. You can reposition objects only within the grid that contains them; they cannot be moved to other grids.

You can also select objects in the Object Browser along the right of the graph. This window shows a hierarchical listing of the objects in the graph. Clicking or right-clicking on an object in the Object Browser is the same as clicking or right-clicking on the object in the graph.

The Graph Editor has the ability to record your actions and play them back on later graphs. When you click on the **Start recording** button, ⬤ , every editing action you take, including undos and redos, is recorded. If you would like to do some editing that is not recorded, you can click on the **Pause recording** button, ❚❚ . You can click on the **Pause recording** button again to resume recording. When you are done with your recording, click on the **Start recording** button. You will be prompted to save your recording. Any recording you save is available from the **Play recording** button, ▶ , and may be applied to future graphs. You can even play a recording in any Stata graph command by using the play option. See *Graph Recorder* in [G-1] **Graph Editor** for more information.

Stop the editor by selecting **File > Stop Graph Editor** from the main menu or by clicking on the **Graph Editor** button. When you stop the Graph Editor, you will be prompted to save your graph if you have made any changes. If you do not save your graph, your changes will not be lost, but you will risk losing them if you create a new graph in the same Graph window. You must stop the Editor if you would like to work on other tasks in Stata.

Here are a few of the things that you can do with the Editor:
- Add annotations using lines, arrows, and text.
- Add or remove grid lines or reference lines.
- Add or modify titles, captions, and notes.
- Change scatterplots to line plots, connected plots, areas, bars, spikes, or drop lines—and, of course, vice versa.
- Change the size, color, margin, and other properties of your graph's titles (or any other text on the graph).

- Move your legend to another side of the graph, or even place it in the plot region.
- Change the aspect ratio of your graph.
- Stack the bars on a bar graph or turn them into percentages.
- Rotate or change the angle of axis labels.
- Add custom ticks and labels to the axes.
- Change the rule for the number and spacing of ticks and labels on an axis.
- Emphasize a point on the graph, whether marker, bar, spike, or other plot, by making it a custom color, size, or symbol.
- Change the text or properties of a marker label.

Because you can edit every property of every object on the graph, you can change almost anything about your graph. To learn more, see [G-1] **Graph Editor** or type `help graph editor`.

16 Saving and printing results by using logs

Using logs in Stata

When you work on an analysis, it is worthwhile to behave like a bench scientist and keep a lab notebook of your actions so that your work can be easily replicated. Everyone has a feeling of complete omniscience while working intensely—this feeling is wonderful but fleeting. The next day, the exact small details needed for perfect duplication have become obscure. Stata has a lab notebook at hand: the *log* file.

A log file is simply a record of your Results window. It records all commands and all textual output as it happens. Thus it keeps your lab notebook for you as you work. Because it writes the file to disk while it writes the Results window, it also protects you from disastrous failures, be they power failures or computer crashes. We recommend that you start a log file whenever you begin any serious work in Stata.

Logging output

All the output that appears in the Results window can be captured in a log file. Stata can save the file in one of two different formats. By default, Stata will save the file in its Stata Markup and Control Language (SMCL) format, which preserves all the formatting and links from the Results window. You can open these results in the Viewer, and they will behave as though they were in the Results window. If you would rather have plain-text files without any formatting, you can save the file as a plain log file. We recommend using the SMCL format because SMCL files can be translated into a variety of formats readable by applications other than Stata with the **File > Log > Translate...** menu (see [R] **translate**).

To start a log file, click on the **Log** button, 📘 . This will open a standard file dialog that allows you to specify a directory and filename for your log. If you do not specify a file extension, the extension .smcl will be added to the filename. If you specify a file that already exists, you will be asked whether you want to append the new log to the file or overwrite the file with the new log.

Here is an example of a short session:

```
        name:  <unnamed>
         log:  C:\Users\Stata\Documents\Stata\base.smcl
    log type:  smcl
   opened on:  25 Jan 2021, 16:47:40
. sysuse auto
(1978 automobile data)
. by foreign, sort: summarize price mpg

-> foreign = Domestic
    Variable │        Obs        Mean    Std. dev.         Min         Max
    ─────────┼──────────────────────────────────────────────────────────
       price │         52    6072.423    3097.104        3291       15906
         mpg │         52    19.82692    4.743297          12          34

-> foreign = Foreign
    Variable │        Obs        Mean    Std. dev.         Min         Max
    ─────────┼──────────────────────────────────────────────────────────
       price │         22    6384.682    2621.915        3748       12990
         mpg │         22    24.77273    6.611187          14          41
. * be sure to include the above stats in report!
. * now for something completely different
. corr price mpg
(obs=74)
             │      price        mpg
    ─────────┼────────────────────────
       price │     1.0000
         mpg │    -0.4686     1.0000

. log close
        name:  <unnamed>
         log:  C:\Users\Stata\Documents\Stata\base.smcl
    log type:  smcl
   closed on:  25 Jan 2021, 16:47:41
```

There are a few items of interest.

- The header showing the log file's location, type, and starting timestamp is part of the log file. This feature helps when working with multiple log files.
- The two lines starting with asterisks (∗) are comments. Stata ignores the text following the asterisk, so you may type any comment you would like, with any special characters you would like. Commenting is a good way to document your thought process and to mark sections of the log for later use.
- In this example, the log file was closed by using the `log close` command. Doing so is not strictly necessary because log files are automatically closed when you exit Stata.

Stata allows multiple log files to be open at once only if the log files are named. For details on this topic, see `help log`.

Working with logs

Log files are best viewed using Stata's Viewer. Select **File > Log > View...**. If there is a log file open (as shown by the status bar), it will be the default log file to view; otherwise, you need to either type the name of the log file into the dialog or click on the **Browse...** button to find the file with a standard file dialog.

Once you are in the Viewer window, everything behaves as expected: you can copy text and paste between the Viewer and anything else that uses text, such as word processors or text editors. You can even paste into the Command window or the Do-file Editor, but you should take care to copy only commands, not their output. It is okay to copy the prompt ("`.`") at the start of the echoed command because Stata is smart enough to ignore it in the Command window. When working with a word processor, what you paste will be unformatted text; it will look best if you use a fixed-width font, like Courier, to display it.

Viewing your current log file is a good way to keep a reminder of something you have already done or a view of a previous result. The Viewer window takes a snapshot of your log file and hence will not scroll as you keep working in Stata. If you need to see more recent results in the Viewer, click on the **Reload page** button.

For more detailed information about logs, see [U] **15 Saving and printing output—log files** and [R] **log**. For more information about the Viewer, see [GSW] **3 Using the Viewer**.

Printing logs

To print a standard SMCL log file, you need to have the log file open in a Viewer window. Once the log file is in the Viewer, you can click on the **Print** button, right-click on the Viewer window, and select **Print...**, or select **File > Print...**. A *Print* dialog will appear. After you click on **Print**, a *Print settings* dialog will appear.

- You can fill in none of, any of, or all the items *Header*, *Name*, and *Project*. You can check or uncheck options to *Print line numbers*, *Print header*, and *Print logo*. These items are saved and will appear again in the print sheet *Print settings* (in this and in future Stata sessions).

- You can set the font, margins, and color scheme that the printer will use by clicking on **Prefs...** in the *Print settings* dialog to open the *Printer preferences* dialog. *Monochrome* is for black-and-white printing, *Color* is for default color printing, and *Custom 1* and *Custom 2* are for customized color printing. You can set the font by clicking on the **Font...** button. The resulting *Font* dialog will list only the fixed-width "typewriter" fonts (for example, Courier) available for your printer.

You could also use the `translate` command to generate a PostScript or PDF version of the log file. See [R] **translate** for more information.

If your log file is a plain-text file (`.log` instead of `.smcl`), you can open it in a text editor, such as Notepad, in the Do-file Editor or in your favorite word processor. You can then edit the log file—add headings, comments, etc.—format it, and print it. If you bring the log file into a word processor, it will be displayed and printed with its default font. The log file will not be easily readable when printed in a proportionally spaced font (for example, Times New Roman or Georgia). It will look much better printed in a fixed-width font (for example, Courier New).

You may wish to associate the `.log` extension with a text editor (such as Notepad or WordPad) in Windows. You can then edit and print the logs from those Windows applications if you like.

Rerunning commands as do-files

Stata also can log just the commands from a session without recording the output. This feature is a convenient way to make a do-file interactively. Such a file is called a *cmdlog* file by Stata. You can start a `cmdlog` file by typing

    `cmdlog using` *filename*`.do`

and you can close the `cmdlog` file by typing

    `cmdlog close`

Here, for example, is what a `cmdlog` of the previous session would look like. It contains only commands and comments and hence could be used as a do-file.

```
sysuse auto
by foreign, sort: summarize price mpg
* be sure to include the above stats in report!
* now for something completely different
corr price mpg
```

If you start working and then wish you had started a `cmdlog` file, you can save yourself heartache by saving the contents of the History window. The History window stores the last 5,000 commands you have typed. Simply right-click on the History window and select **Save all...** from the menu. This will work best if you first filter out all the commands that resulted in errors as was shown in *The History window* in [GSW] **2 The Stata user interface**. If you would like to move the commands directly to the Do-file Editor, select **Select all** followed by **Send selected to Do-file Editor**. You may find this method a more convenient way to create a text file containing only the commands that you typed during your session.

See [GSW] **13 Using the Do-file Editor—automating Stata**, [U] **16 Do-files**, and [U] **15 Saving and printing output—log files** for more information.

17 Setting font and window preferences

Changing and saving fonts and sizes and positions of your windows

You may find that you would like to change the fonts and display style of Stata's windows, depending on your monitor resolution and personal preferences. At the same time, there could be requirements for font usage, say, when you submit graphs to journals. Stata accommodates both of these by allowing sets of preferences for how windows are displayed.

We will first cover what can be changed in each window and then talk about what you can manage with your preferences.

Graph window

The preferences for the Graph window can be changed by right-clicking on the Graph window and choosing **Preferences...** from the contextual menu. The settings can then be set for how graphs are displayed in Stata. The settings that should be used when printing can be set under the **Printer** tab. The behavior of the Clipboard is controlled under the **Clipboard** tab.

The Graph preferences allow different schemes that control the look of graphs. These schemes provide a quick way to optimize graphs for printing or to display on a screen. There are even schemes defined for *The Economist* and the *Stata Journal* so that you can get the details for these publications right without much fuss. Changing the scheme does not change the current graph—it applies the settings to future graphs.

All other windows

You can change the display font and font size for most types of windows in Stata.

If fonts and font sizes for a window can be changed, they can be changed by right-clicking on the window and selecting **Font...** from the contextual menu. Doing so will bring up the *Font* dialog, from which you can pick the font and size of your choice. The font lists for each of the Results, Viewer, Data Editor, and Do-file Editor windows are restricted to fixed-width fonts only. This restriction ensures that output and numbers line up properly and are readable. The other windows can have any font that you would like without any adverse consequences.

Changing color schemes

In addition to changing the fonts themselves, you can also change the background and foreground colors of text being displayed. In the Do-file Editor, you can choose the colors for syntax highlighting, allowing, say, Stata commands to be displayed in a different color from arbitrary text. You can control the overall color scheme by selecting a scheme in the **General** tab of the General preferences.

The Results and Viewer windows have color schemes that control the display of input, text, results, errors, links, and highlighted text. Each has its color scheme set in the same fashion: you can right-click on the window and select or design your own color scheme. The default setting for both the Results window and the Viewer is the built-in *Standard* scheme, which uses a white background and dark text. There are other built-in schemes as well as slots for custom schemes. The settings for the Viewer affect all Viewer windows at once. Choosing an overall scheme from the **General** tab will reset all custom settings to the settings determined by that scheme.

Managing multiple sets of preferences

Stata's preferences are automatically saved when you exit Stata, and they are reloaded when Stata is launched. However, sometimes you may wish to rearrange Stata's windows and then revert to your preferred arrangement of windows. You can do this by saving your preferences to a named preference set and loading them later. Any changes you make to Stata's preferences after loading a preferences set do not affect the set; the set remains untouched unless you specifically overwrite it.

To manage preferences, open the **Edit > Preferences** menu, and do any of the following:

- Select a preference set from the **Load preference set** menu to load it. Several different preference sets come installed with Stata, some meant for small screens, others meant for giving presentations involving Stata. They are worth a look.
- Select **Save preference set > New preference set...** to save the current preferences to a set. Enter a name for the set, and click on **OK**.
- Select an existing set from the **Save preference set** menu to overwrite it with the current preferences.
- Select a preference set from the **Delete preference set** menu to delete it. Click on **OK** to verify that you wish to delete the set.

Closing and opening windows

You can close all windows but the Results and Command windows. If you want to open a closed window, open the **Window** menu and select the desired window.

18 Learning more about Stata

Where to go from here

You now know plenty enough to use Stata. There is still much, much more to learn because Stata is a rich environment for doing statistical analysis and data management. What should you do to learn more?

- Get an interesting dataset and play with Stata.
    a. Use the menus and dialog system to experiment with commands. Notice what commands show up in the Results window. You will find that Stata's simple and consistent command syntax will make the commands easy to read so that you will know what you have done and easy to remember so that typing some commands will be faster than using menus.
    b. Play with graphs and the Graph Editor.
- If you venture into the Command window, you will find that many things will go faster. You will also find that it is possible to make mistakes where you cannot understand why Stata is balking.
    a. Try `help` *commandname* or **Help > Stata command...** and entering the command name.
    b. Look at the command syntax and the examples in the help file, and compare them with what you typed. Compare them closely: small typographical errors make commands impossible for Stata to parse.
- Explore Stata by selecting **Help > Search...**. You will uncover many statistical routines that could be of great use.
- Look through the *Combined subject table of contents* in the *Stata Index*.
- Read and work your way through the *User's Guide*. It is designed to be read from cover to cover, and it contains most of the information you need to become an expert Stata user. It is well worth reading. If you are not this ambitious and instead prefer to sample the *User's Guide* and the references, there is some advice later in this chapter for you.
- Browse through the reference manuals to read about statistical methods you like to use, making use of the links to jump to other topics. The reference manuals are not meant to be read from cover to cover—they are meant to be referred to as you would an encyclopedia. You can find the datasets used in the examples in the manuals by selecting **File > Example datasets...** and then clicking on `Stata 17 manual datasets`. Doing so will enable you to work through the examples quickly.
- Stata has much information, including answers to frequently asked questions (FAQs), at https://www.stata.com/support/faqs/.
- There are many useful links to Stata resources at https://www.stata.com/links/. Be sure to look at these materials because many outstanding resources about Stata are listed here.
- Join Statalist, a forum devoted to discussion of Stata and statistics.
- Read *The Stata Blog: Not Elsewhere Classified* at https://blog.stata.com to read articles written by people at Stata about all things Stata.
- Visit Stata on Facebook at https://facebook.com/statacorp,
  join Stata on Instagram at https://www.instagram.com/statacorp,
  find Stata on LinkedIn at https://www.linkedin.com/company/statacorp,
  and follow Stata on Twitter at https://twitter.com/stata to keep up with Stata.
- Subscribe to the *Stata Journal*, which contains reviewed papers, regular columns, book reviews, and other material of interest to researchers applying statistics in a variety of disciplines. Visit https://www.stata-journal.com.

- Many supplementary books about Stata are available. Visit the Stata Bookstore at https://www.stata.com/bookstore/.
- Take a Stata NetCourse®. NetCourse 101 is an excellent choice for learning about Stata. See https://www.stata.com/netcourse/ for course information and schedules.
- Attend a classroom or a web-based training course taught by StataCorp. Visit https://www.stata.com/training/classroom-and-web/ for course information and schedules.
- View a webinar led by Stata developers. Visit https://www.stata.com/training/webinar/ for the current list of topics and schedule.
- Watch Stata videos at https://www.youtube.com/user/statacorp.

Suggested reading from the User's Guide and reference manuals

The *User's Guide* is designed to be read from cover to cover. The reference manuals are designed as references to be sampled when necessary.

Ideally, after reading this *Getting Started* manual, you should read the *User's Guide* from cover to cover, but you probably want to become at least somewhat proficient in Stata right away. Here is a suggested reading list of sections from the *User's Guide* and the reference manuals to help you on your way to becoming a Stata expert.

This list covers fundamental features and points you to some less obvious features that you might otherwise overlook.

Basic elements of Stata

[U] **11 Language syntax**

[U] **12 Data**

[U] **13 Functions and expressions**

Data management

[U] **6 Managing memory**

[U] **22 Entering and importing data**

[D] **import** — Overview of importing data into Stata

[D] **append** — Append datasets

[D] **merge** — Merge datasets

[D] **compress** — Compress data in memory

[D] **frames intro** — Introduction to frames

Graphics

[G] *Stata Graphics Reference Manual*

Reproducible research

[U] **16 Do-files**

[U] **17 Ado-files**

[U] **13.5 Accessing coefficients and standard errors**

[U] **13.6 Accessing results from Stata commands**

[U] **21 Creating reports**

[RPT] **Dynamic documents intro** — Introduction to dynamic documents

[RPT] **putdocx intro** — Introduction to generating Office Open XML (.docx) files

[RPT] **putexcel** — Export results to an Excel file

[RPT] **putpdf intro** — Introduction to generating PDF files

[R] **log** — Echo copy of session to file

Useful features that you might overlook

[U] **29 Using the Internet to keep up to date**

[U] **19 Immediate commands**

[U] **24 Working with strings**

[U] **25 Working with dates and times**

[U] **26 Working with categorical data and factor variables**

[U] **27 Overview of Stata estimation commands**

[U] **20 Estimation and postestimation commands**

[R] **estimates** — Save and manipulate estimation results

Basic statistics

[R] **anova** — Analysis of variance and covariance

[R] **ci** — Confidence intervals for means, proportions, and variances

[R] **correlate** — Correlations of variables

[D] **egen** — Extensions to generate

[R] **regress** — Linear regression

[R] **predict** — Obtain predictions, residuals, etc., after estimation

[R] **regress postestimation** — Postestimation tools for regress

[R] **test** — Test linear hypotheses after estimation

[R] **summarize** — Summary statistics

[R] **table intro** — Introduction to tables of frequencies, summaries, and command results

[R] **tabulate oneway** — One-way table of frequencies

[R] **tabulate twoway** — Two-way table of frequencies

[R] **ttest** — t tests (mean-comparison tests)

Matrices

[U] **14 Matrix expressions**

[U] **18.5 Scalars and matrices**

[M] *Mata Reference Manual*

Programming

[U] **16 Do-files**

[U] **17 Ado-files**

[U] **18 Programming Stata**

[R] **ml** — Maximum likelihood estimation

[P] *Stata Programming Reference Manual*

[M] *Mata Reference Manual*

System values

[R] **set** — Overview of system parameters

[P] **creturn** — Return c-class values

Internet resources

The Stata website (https://www.stata.com) is a good place to get more information about Stata. You will find answers to FAQs, ways to interact with other users, official Stata updates, and other useful information. You can also join Statalist, a forum devoted to discussion of Stata and statistics.

You will also find information on Stata NetCourses®, which are interactive courses offered over the Internet that vary in length from a few weeks to eight weeks. Stata also offers in-person and web-based training sessions, as well as webinars on Stata features. Visit https://www.stata.com/learn/ for more information.

At the website is the Stata Bookstore, which contains books that we feel may be of interest to Stata users. Each book has a brief description written by a member of our technical staff explaining why we think this book may be of interest.

We suggest that you take a quick look at the Stata website now. You can register your copy of Stata online and request a free subscription to the *Stata News*.

Visit https://www.stata-press.com for information on books, manuals, and journals published by Stata Press. The datasets used in examples in the Stata manuals are available from the Stata Press website.

Also visit https://www.stata-journal.com to read about the *Stata Journal*, a quarterly publication containing articles about statistics, data analysis, teaching methods, and effective use of Stata's language.

Visit Stata's official blog at https://blog.stata.com for 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. *The Stata Blog: Not Elsewhere Classified* also has links to other blogs about Stata, written by Stata users around the world.

Follow Stata on Facebook at https://facebook.com/statacorp, Twitter at https://twitter.com/stata, Instagram at https://www.instagram.com/statacorp, and LinkedIn at https://www.linkedin.com/company/statacorp. You may also follow Stata on Twitter at https://twitter.com/stata_fr or https://twitter.com/stata_es. These are good ways to stay up-to-the-minute with the latest Stata information. Watch short example videos of using Stata on YouTube at https://www.youtube.com/user/statacorp.

See [GSW] **19 Updating and extending Stata—Internet functionality** for details on accessing official Stata updates and free additions to Stata on the Stata website.

19 Updating and extending Stata—Internet functionality

Internet functionality in Stata

Stata works well with the Internet. Stata can use datasets and view remote help files as though they were on your computer. Stata also can keep itself up to date (with your permission, of course). Finally, you can install community-contributed commands, which are commands that extend Stata's functionality. These are commands that have been presented in the *Stata Journal* (SJ) or the *Stata Technical Bulletin* (STB) or have simply been written and shared by the greater Stata community.

This chapter will show you how you can expand Stata's horizons.

Using files from the Internet

Stata understands URLs as though they were local file locations. If you know of a file on the web that you would like to use, be it a dataset, a graph, or a do-file, you can easily open it in Stata. Here is a small example.

There are many datasets at https://www.stata-press.com/data/. Suppose that you would like to use the `census12` dataset used in [U] **11 Language syntax** and that you know that its location is https://www.stata-press.com/data/r17/census12.dta. Because you know that the command for opening a dataset is `use`, you could type the following:

```
. use https://www.stata-press.com/data/r17/census12.dta
(1980 Census data by state)

. describe

Contains data from https://www.stata-press.com/data/r17/census12.dta
 Observations:            50                  1980 Census data by state
    Variables:             7                  6 Apr 2020 15:43
─────────────────────────────────────────────────────────────────────
Variable      Storage   Display    Value
    name         type    format    label      Variable label
─────────────────────────────────────────────────────────────────────
state         str14     %14s                  State
state2        str2      %-2s                   Two-letter state abbreviation
region        str7      %9s                    Census region
pop           long      %10.0g                 Population
median_age    float     %9.2f                  Median age
marriage_rate float     %9.0g
divorce_rate  float     %9.0g
─────────────────────────────────────────────────────────────────────
Sorted by:
```

This functionality is everywhere in Stata. Any command that reads a file with a *filename* in its syntax can use a web address as easily as a file that is stored on your computer.

This example used the HTTP protocol for retrieving the file. Stata also understands the HTTPS and FTP protocols.

Official Stata updates

By official Stata, we mean the pieces of Stata that are provided and supported by StataCorp. The other and equally important pieces are the community-contributed additions published in the sj, distributed over Statalist, or distributed in other ways.

Stata can fetch both official updates and community-contributed commands from the Internet. Let's start with the official updates. StataCorp often releases updates to official Stata. These updates add new features and, sometimes, fix bugs.

By default, Stata has automatic update checking turned on and set to check for updates every seven days. To change or check your settings, select **Edit > Preferences > General preferences...** and click on the **General** tab.

We recommend using automatic update checking because it is a simple, unobtrusive way to be sure that your copy of Stata is always up to date. If you keep this default, you will be prompted with a dialog when you start Stata if you have not recently checked for updates.

To manually check whether there are any official Stata updates, either click on **Help > Check for updates** or type `update query` in the Command window. Regardless of which choice you make, Stata goes to check for official updates. After it checks, it will show you your update status. If your copy of Stata is already up to date, you will be told. If your copy of Stata needs updating, you will be told, and a link, `Install available updates`, will show up in your Results window. You can click on this link or type `update all` and press *Enter*. In either case, Stata will download what is needed to bring your copy of Stata up to date. Stata will need to restart after being updated, so it gives you a chance to postpone the update in case there was something (such as saving the command history) you wanted to do in the current session.

Troubleshooting note: If you do not have write permission for `C:\Program Files\Stata17`, you cannot install official updates in this way. You may still download the official updates, but you will need to use the command-line version of `update`; see [U] **29 Using the Internet to keep up to date** for instructions.

Automatic update checking

Stata can periodically check for updates for you. By default, Stata will check once every seven days for updates from the StataCorp website. The seven-day interval is from the last time an `update query` was performed, regardless of whether it was by Stata or by you. You can change the interval between checks.

Before Stata connects to the Internet to check for an update, it will ask you if you would like to check now, check the next time Stata is launched, or check after the next interval. You can disable the prompt and allow Stata to check without asking.

If an update is available, Stata will notify you. From there, you should follow the recommendations for updating Stata.

You can change the settings for automatic update checking by selecting **Edit > Preferences > General preferences...** and choosing **General**.

Finding community-contributed commands by keyword

Stata has a built-in utility created specifically to search the Internet for community-contributed Stata commands. You can access it by selecting **Help > Search...**, choosing *Search net resources*, and entering a keyword in the field. Choosing **Help > SJ and community-contributed features** yields more specific choices for searching. The utility searches all community-contributed commands on the Internet, including the entire collection of SJ and STB commands. The results are displayed in the Viewer, and you can click to go to any of the matches found.

For the syntax on how to use the equivalent `search` *keywords*, `net` command, see [R] **search**.

Downloading community-contributed commands

Downloading community-contributed commands is easy. Start by selecting **Help > SJ and community-contributed features**:

```
  Viewer - help net_mnu                                    —  □  ✕

  File   Edit   History   Help

  ←  →  C  🖶  Q   help net_mnu                Q▾

  help net_mnu   ✕                                                 ▾

  ✚                                        Dialog ▾  Also see ▾  Jump to ▾

  Installation and maintenance of SJ, STB, and community-contributed programs

        Community-contributed programs -- SJ, STB, Statalist, and others -- are
        available from a variety of sources.  Use the links below to find, install,
        and uninstall them.

  New packages

        Search...              Search all sources.  Try this first!

        Stata Journal          Programs from Stata Journal articles
        STB                    Programs from Stata Technical Bulletin articles
        SSC archive            Boston College Statistical Software Components
                                     Stata program archive
        Other locations        Other locations with available programs

                                                    CAP  NUM  OVR
```

As the Viewer says, try `Search...` first.

Suppose that you were interested in finding more information or some community-contributed commands involving goodness of fit for logistic regression. You select **Help > Search...**, select *Search all*, type `logistic goodness of fit` in the search box, and click on the **OK** button.

Viewer - search logistic goodness of fit   — □ ✕

File   Edit   History   Help

← → C 🖶 Q   search logistic goodness of fit   Q▾ [   ]

search logistic goodness of fit   ✕   ▼

+   Dialog ▾ Also see ▾ Jump to ▾

---

search for **logistic goodness of fit**   (manual: [R] search)

Search of official help files, FAQs, Examples, and Stata Journals

[R]   logistic postestimation . . . . . . Postestimation tools for logistic
       (help logistic postestimation)

[R]   estat gof . . . . . . Pearson or Hosmer-Lemeshow goodness-of-fit test
       (help logistic estat gof)

FAQ   . . . . What statistical analysis should I use?
       . . . . UCLA Academic Technology Services
       5/08   https://stats.idre.ucla.edu/stata/whatstat/what-statistical-
               analysis-should-i-usestatistical-analyses-using-stata/

Example . . . . Textbook examples: Applied Logistic Regression (2nd Edition)
       . . . . UCLA Academic Technology Services
       2/08   examples from the book **Applied Logistic Regression
               (2nd Edition)** by David Hosmer and Stanley Lemeshow
               https://stats.idre.ucla.edu/other/examples/alr2/

Example . . . . Seminar: Logistic regression with Stata
       . . . . UCLA Academic Technology Services
       5/07   seminar to help increase skills using logistic
               regression analysis
               https://stats.idre.ucla.edu/stata/seminars/stata-logistic/

SJ-17-4 gr0071 . Calibration of dichot. outcome models with calibration belt
               . . G. Nattino, S. Lemeshow, G. Phillips, S. Finazzi, and G. Bertolini
               (help calibrationbelt if installed)
               Q4/17   SJ 17(4):1003--1014
               implements the calibration belt (a graphical approach to
               evaluate the goodness of fit of binary outcome models by
               examining the relationship between estimated probabilities
               and observed outcome rates) and its associated test

SJ-13-2 st0299 . . . . Goodness-of-fit tests for categorical data
               . . . . R. Bellocco and S. Algeri
               Q2/13   SJ 13(2):356--365   (no commands)
               discusses choice of analytical units of reference (subjects
               or groups of subjects that have the same covariate pattern)
               and how that affects the definition of the saturated model
               and conclusions from goodness-of-fit tests

SJ-12-3 st0269 . . . . . . A generalized Hosmer-Lemeshow goodness-of-fit test
               (help mlogitgof if installed) . . . . M. W. Fagerland and D. W. Hosmer
               Q3/12   SJ 12(3):447--453
               implements a generalized Hosmer-Lemeshow goodness-of-fit test
               for multinomial logistic regression models

                                            CAP   NUM   INS

The first entry points you to all the postestimation commands that are available after logistic regression. The second entry points to Stata's built-in `estat gof` command specifically for computing goodness-of-fit statistics after logistic regression. You investigate this command and find it interesting. You see that the next three links point to FAQs and examples on UCLA's website. Then the next three links are for articles in the SJ. You are interested in multinomial logistic regression, so you decide to check the last of these links. It points to an article in the SJ, volume 12, number 3 (third quarter). You should click on the `st0269` link because it will go to the command associated with this article.

You will see that the package has one help file for the new commands. Click the `st0269/mlogitgof.sthlp` link to see if the `mlogitgof` command looks interesting. If you decide that you would like to install the command, click the **Back** button and click on the link `click here to install`. If you decide that you would like to use some of the ancillary files—files that typically help explain the workings of the command, you could download those, too. You do not need to worry—doing so will not interfere in any way with your copy of Stata. We will show you how to safely uninstall these commands shortly.

You can keep the community-contributed commands you have installed up to date by using the `ado update` command. Typing `ado update` will check for updates, while typing `ado update, update` will check for updates and install any available updates.

Now suppose that you decide that you would like to uninstall the package. Doing so is simple enough: select **Help > SJ and community-contributed features**, and click on the `List` link. You should see the following:

If you click on the one-line description of the package, you will see the full description of what has been installed. Here is what you would see if you scroll to the bottom, with a different install date, of course:

```
 Viewer - ado describe                                          —    □    ✕

 File   Edit   History   Help

 ←  →  C  🖶  🔍  | ado describe                     |      Q▾ |          |

  ado describe   ✕                                                        ▾

  ✚                                         Dialog ▾ | Also see ▾ | Jump to ▾
                                                                          ⌃

     ─────────────────────────────────────────────────────────────────
     [1] package st0269 from http://www.stata-journal.com/software/sj12-3
     ─────────────────────────────────────────────────────────────────

     TITLE                                        (click here to uninstall)
           SJ12-3 st0269.   A generalized Hosmer-Lemeshow...

     DESCRIPTION/AUTHOR(S)
           A generalized Hosmer-Lemeshow goodness-of-fit test for
             multinomial logistic regression models
           by Morten W. Fagerland, Unit of Biostatistics and
                 Epidemiology, Oslo University Hospital, Norway
              David W. Hosmer, Department of Public Health,
                 University of Massachusetts, Amherst, Amherst, MA
           Support:  morten.fagerland@medisin.uio.no
           After installation, type help mlogitgof

     INSTALLATION FILES
           m\mlogitgof.ado
           m\mlogitgof.sthlp

     INSTALLED ON
           19 Feb 2021
     ─────────────────────────────────────────────────────────────────
     (click here to return to the previous screen)
                                                                          ⌄

                                                        CAP   NUM   INS
```

You can uninstall materials by clicking on `click here to uninstall` when you are looking at the package description.

For information on downloading community-contributed commands by using the `net` command, see [R] **net**.

A   Troubleshooting Stata

Contents

A.1 If Stata does not start

You tried to start Stata and it refused; Stata or your operating system presented a message explaining that something is wrong. Here are the possibilities:

Cannot find license file

This message means just what it says; nothing is too seriously wrong. Stata simply could not find the license file it was looking for. The most common reason for this is that you did not complete the installation process.

Did you enter the codes on your license to unlock Stata? If not, go back and complete the initialization procedure.

Error opening or reading the file

Something is distinctly wrong for purely technical reasons. Stata found the file that it was looking for, but there was an I/O error.

About the only way this situation could arise would be a hard-disk error. Stata technical support will be able to help you diagnose the problem; see [U] **3.8 Technical support**.

License not applicable

Stata has determined that you have a valid Stata license, but the license does not apply to the version of Stata that you are trying to run.

The most common reason for this message is that you have a license for Stata/BE but you are trying to run Stata/SE or Stata/MP, or you have a license for Stata/SE but you are trying to run Stata/MP. If any of these is the case, insert the installation DVD, run the installer again, choose *Modify*, click on the **Next** button, and choose the appropriate edition.

Other messages

The other messages indicate that Stata thinks you are attempting to do something that you are not licensed to do. Most commonly, you are attempting to run Stata over a network when you do not have a network license, but there are many other alternatives. There are two possibilities: either you really are attempting to do something that you are not licensed to do or Stata is wrong. In either case, you are going to have to contact us. Your license can be upgraded, or if Stata is wrong, we can provide codes to make Stata stop thinking that you are violating the license; see [U] **3.8 Technical support**.

A.2 Troubleshooting tips

If you experience an unexpected problem, first make sure that you are running the most current version of Stata (see [GSW] **19 Updating and extending Stata—Internet functionality** for information on updating). If the problem still exists, look at the frequently asked questions (FAQs) for Windows in the user-support section of the Stata website, https://www.stata.com/support/faqs/windows/. You may find the answer to the problem there. If not, we can help, but you must give us as much information as possible.

Reboot your computer, restart Stata, and try to reproduce the problem, writing down everything you do before the fault occurs. We will want that information.

If Stata used to work on your computer but suddenly stopped working, try to remember any hardware or software that you have recently installed.

Also give us as much information about your computer as possible. What version of Windows are you running? How much memory do you have? What processor do you have? What brand is your computer? Finally, we need your Stata serial number and the revision date of your version of Stata. Include them if you email, and know them if you call. You can obtain them by typing the `about` command in Stata's Command window. `about` lets you know everything about your copy of Stata, including the version and the date it was produced.

B Advanced Stata usage

Contents

B.1 The Windows Properties Sheet

When you double-click on a shortcut to start an application in Windows, you are actually executing instructions defined in the shortcut's Properties Sheet. To open the Properties Sheet for any shortcut, right-click on the shortcut, and select **Properties**.

Open the Properties Sheet for Stata's shortcut. Click on the **Shortcut** tab. You will see something like the following:

StataSE

Target type:	Application
Target location:	Stata17
Target:	`C:\Program Files\Stata17\StataSE.exe /UseRegistryStartin`
Start in:	
Shortcut key:	`None`
Run:	`Normal window`

The field names may be slightly different, depending on the version of Windows that you are running. The names and locations of files may vary from this. There are two things to pay attention to: the *Target* and *Start in* fields. *Target* is the actual command that is executed to invoke Stata. *Start in* is the directory to switch to before invoking the application. You can change these fields and then click on **OK** to save the updated Properties Sheet.

You can have Stata start in any directory you desire. If necessary, delete the parameter /UseRegistryStartin from your *Target* field. Then change the *Start in* field of Stata's Properties Sheet to the location you would like Stata to have as its default working directory. Of course, once Stata is running, you can change directories whenever you wish by using **File > Change working directory...**; see also [D] **cd**.

B.2 Making shortcuts

You can arrange to start Stata without going through the **Start** menu by creating a shortcut on the Desktop. The easiest way to do this is to copy the existing Stata shortcut to the Desktop. You can also create a shortcut directly from the Stata executable. Here are the details:

1. Open the `C:\Program Files\Stata17` folder or the folder where you installed Stata.

2. In the folder, find the executable for which you want a new shortcut. The filenames for the 64-bit versions of Stata are

Stata/MP:	StataMP-64.exe
Stata/SE:	StataSE-64.exe
Stata/BE:	Stata-64.exe

   Right-click on and drag the appropriate executable onto the Desktop.

3. Release the mouse button, and select **Create Shortcut(s) Here** from the menu that appears.

You have now created a shortcut. If you want the shortcut in a folder rather than on the Desktop, you can drag it into whatever folder appeals to you.

You set the properties for this shortcut just as you would normally. Right-click on the shortcut, and select **Properties**. Edit the Properties Sheet as explained above in [GSW] **B.1 The Windows Properties Sheet**.

B.3 Executing commands every time Stata is started

Stata looks for the file `profile.do` when it is invoked and, if it finds it, executes the commands in it. Stata looks for `profile.do` first in the directory where Stata is installed, then in the current directory, then along your path, then in your home directory as defined by Windows's USERPROFILE environment variable (typically `C:\Users\`*username*), and finally along the ado-path (see [P] **sysdir**). We recommend that you put `profile.do` in your home directory.

If you create a shortcut that starts in a different directory, it will run the `profile.do` from that directory. This feature allows you to have different `profile.do` files for different projects.

Say that every time you start Stata, you would like to start a dated log for the session. In your default working directory, say, `C:\Users\Stata\Documents\Stata`, create the file `profile.do` containing this rather odd-looking command:

```
log using `: display %tCCCYY-NN-DD-HH-MM-SS ///
   Clock("`c(current_date)' `c(current_time)'","DMYhms")', ///
   name(default_log_file)
```

When you invoke Stata, the usual opening appears but with the following additional command, which will be executed:

```
running C:\Users\Stata\Documents\Stata\profile.do ...
```

How does the command work? Let's work from the inside out:
- `c(current_date)` and `c(current_time)` are local system macros containing the current date and current time. See [P] **creturn** for more information.
- The left (`) and right (') quotes around the local macros expand them. See [P] **macro** for a full explanation.
- The `Clock()` function uses the resulting date string and the date mask `"DMYhms"` to create a datetime number Stata understands. See [D] **Datetime**.

- The format %tCCCYY-NN-DD-HH-MM-SS formats this number in year-month-day-hour-minute-second form because this will make the files sort nicely. See [D] **Datetime display formats** for the details.
- The odd-looking ': display ...' allows the formatted date to be used directly in the command as the file name. This is the advanced concept of an in-line expansion of a macro function. You can see more in [P] **macro**.
- The log using command starts a log file, such as shown in [GSW] **16 Saving and printing results by using logs**.
- The name option gives the log file the internal name default_log_file so that it will not likely conflict with other log files. See [R] **log** for details.
- Finally, the /// notations are continuation comments so that the three separate lines are interpreted as a single command. See [P] **comments** for more about comments.

There are many advanced Stata programming concepts in this one single command!

profile.do is treated just as any other do-file once it is executed; results are just the same as if you had started Stata and then typed run profile.do. The only special thing about profile.do is that Stata looks for it and runs it automatically.

System administrators might also find sysprofile.do useful. This file is handled in the same way as profile.do, except that Stata first looks for sysprofile.do. If that file is found, Stata will execute any commands it contains. After that, Stata will look for profile.do and, if that file is found, execute the commands in it.

One example of how sysprofile.do might be useful would be when system administrators want to change the path to one of Stata's system directories. Here sysprofile.do could be created to contain the command

```
sysdir set SITE "\\Matador\StataFiles"
```

See [U] **16 Do-files** for an explanation of do-files. They are nothing more than text files containing sequences of commands for Stata to execute.

B.4 Other ways to launch Stata

The first time that you start Stata for Windows, Stata registers with Windows the actions to perform when you double-click on certain types of files. You can then start a new instance of Stata by double-clicking on a Stata .dta dataset, a Stata .do do-file, or a Stata .gph graph file. In all cases, your current working directory will become the folder containing the file you have double-clicked.

Stata will behave as you would expect in each case. If you double-click on a dataset, Stata will open the dataset after Stata starts. If you double-click on a graph, the graph will be opened by Stata. If you double-click on a do-file, the do-file will be opened in the Do-file Editor.

If you would rather run a do-file directly, right-click on the do-file. You will see menu items for **Execute (do)** and **Execute quietly (run)**. These items will complete the requested action in a new instance of Stata.

If you want to edit a do-file, look at a graph, or open a dataset without starting a new instance of Stata, drag the file over Stata's main window.

B.5 Stata batch mode

You can run large jobs in Stata in batch mode. There are a few different ways to do this, depending on what your goals are.

Method 1

If you have a particular location where your log file should be after the job is done, this is the method you should use.

In Windows 10, type `cmd` in the search box in the taskbar, and press *Enter*.

In Windows 8, point your mouse at the hotspot in the right corner of the screen to access the Charms Sidebar, select the search charm, and start typing `command` in the search field. Once the list of possible applications shrinks down to just the **Command Prompt** app, press *Enter*. In older versions of Windows, click on the **Start** menu and choose **Run...**. If you are working on Windows 7 and do not see the **Run...** menu item, then you need to click on **Start > All Programs > Accessories > Run**. In the *Open* field, type

```
cmd
```

and press *Enter*.

You should now have a command prompt window open. Change the current directory to the place you would like the log file to be by using the `cd` command. For example, suppose your `bigjob` do-file is in C:\Users\someone\statastuff, and you would like to save your log in C:\Users\someone\statalogs. You would type the following to suppress all screen output and place the log file in the proper location:

```
cd C:\Users\someone\statalogs
"C:\Program Files\Stata17\StataSE.exe" /e do "C:\Users\someone\statastuff\bigjob"
```

You must specify the location of the Stata executable.

The /e parameter above tells Stata how to behave when running in batch mode. The available parameters and their purposes are

Parameter	Result
/b	set background (batch) mode and log in plain text
/e	set background (batch) mode and log in plain text without prompting when Stata command has completed
/q	suppress logo and initialization messages
/rngstream#	set random-number generator to mt64s (see [R] **set rng**) and set random-number stream to # (see [R] **set rngstream**)
/s	set background (batch) mode and log in SMCL

Method 2

If you would like to have a batch job that you could run at a particular time or that you could save for later use, you can use the Task Scheduler, which is part of most Windows installations.

This is a bit more advanced, and its implementation differs slightly for each kind of Windows, but here is the general gist.

In Windows 10, you can search for the Task Scheduler in the search box in the taskbar. In Windows 8, you can search for the Task Scheduler by using the search charm. In earlier versions of Windows, you can generally find the Task Scheduler by clicking **Start > All Programs > Accessories > System Tools > Task Scheduler**. Once you have opened the Task Scheduler, click on **Create Basic Task**, and follow the steps of the Basic Task Schedule Wizard to schedule a do-file to run in batch mode. You must specify the /b or /e option. In the *Start in* field, type the path where you would like the log file to be saved. When this file runs, all output will be suppressed and written to a log file that will be saved in the path specified.

General notes

While your do-file is executing, the Stata icon will appear on the taskbar.

If you click on the icon on the taskbar, Stata will display a box asking if you want to cancel the batch job.

Once the do-file is complete, Stata will flash the icon on the taskbar on and off. You can then click on the icon to close Stata. If you wish for Stata to automatically exit after running the batch do-file, use /e rather than /b.

You do not have to run large do-files in batch mode. Any do-file that you run in batch mode can also be run interactively. Simply start Stata, type log using *filename*, and type do *filename*. You can then watch the do-file run, or you can minimize Stata while the do-file is running.

B.6 Running simultaneous Stata sessions

Each time you double-click on the Stata icon or launch Stata in any other way, you invoke a new instance of Stata, so if you want to run multiple Stata sessions simultaneously, you may. The title bar of each new Stata that is invoked will reflect its instance number.

B.7 Changing Stata's locale

To change the locale of Stata to English, type

```
set locale_ui en
```

To change it back to match the locale set for your operating system, type

```
set locale_ui default
```

For a complete explanation of locales and Stata, see [U] **12.4.2.4 Locales in Unicode**.

B.8 More

If you would like Stata to pause every time the screen fills with results, type `set more on`. This will cause a —more— prompt to appear at the bottom of the Results window whenever there is more information to be displayed than can fit on the screen. This happens, for example, when you are `listing` many observations.

```
. list make mpg

        make              mpg

  1.    Linc. Continental  12
  2.    Linc. Mark V       12
  3.    Cad. Deville       14
  4.    Cad. Eldorado      14
  5.    Linc. Versailles   14

  6.    Merc. Cougar       14
  7.    Merc. XR-7         14
  8.    Peugeot 604        14
  9.    Buick Electra      15
 10.    Merc. Marquis      15

 11.    Buick Riviera      16
 12.    Chev. Impala       16
 13.    Dodge Magnum       16
 14.    Olds Toronado      16
 15.    AMC Pacer          17

 16.    Audi 5000          17
 17.    Dodge St. Regis    17
 18.    Volvo 260          17
 19.    Buick LeSabre      18
 20.    Dodge Diplomat     18

—more—
```

If you want to see the next screen of text, you have a few options: press any key, such as the Spacebar; click on the **More** button, ⬇ ▾ ; or click on the —more— link at the bottom of the Results window. To see just the next line of text, press *Enter*. Pressing `q` will interrupt the command. If you click on the arrow of the **More** button, you can also select the **Run to completion** menu item to let the command completely finish.

B.9 Memory size considerations

Memory management in Stata is automatic. For details on efficiency tweaks needed by a very few Stata users, look at [D] **memory**.

C   More on Stata for Windows

C.1 Using Stata datasets and graphs created on other platforms

Stata will open any Stata `.dta` dataset or `.gph` graph file regardless of the platform on which it was created, even if it was a Mac or Unix system. Also Stata for Mac and Stata for Unix users can use any files that you create. If you transfer a Stata file by using file transfer protocol (FTP), just remember to transfer by using binary mode rather than ASCII.

C.2 Exporting a Stata graph to another document

Suppose that you wish to export a Stata graph to a document created by your favorite word processor or presentation application. You have two main choices for exporting graphs: you may copy and paste the graph by using the Clipboard, or you may save the graph in one of several formats and import the graph into the application.

C.2.1   Exporting the graph by using the Clipboard

The easiest way to export a Stata graph into another application is by copying and pasting.

Either create your graph or redisplay an existing graph. To copy it to the Clipboard, right-click on the Graph window, and select **Copy**. Stata will copy the graph as an Enhanced Metafile (EMF); this ensures that the receiving application obtains it in the highest resolution possible.

A metafile contains the commands necessary to redraw the graph. That is, a metafile is a collection of lines, points, text, and color information. Metafiles, therefore, can be edited in a structured drawing program.

After you have copied the graph to the Clipboard, switch to the application into which you wish to import the graph and paste it. In most applications, this is accomplished by selecting **Edit > Paste**. Consult the documentation for your particular application for more details.

C.2.2 Exporting the graph to a file

Stata can export graphs to several different file formats. If you right-click on a graph, select **Save as...**, and then click on the drop-down menu next to *Save as type*, you will see that Stata can save in the following file types:

Enhanced Metafile (EMF, .emf), Enhanced PostScript (EPS, .eps) with or without a TIFF Preview, Joint Photographic Experts Group (JPEG, .jpg) with High Quality or Maximum Quality, Portable Document Format (PDF, .pdf), Portable Network Graphics (PNG, .png), PostScript (PS, .ps), Scalable Vector Graphics (SVG, .svg), and Tag Image File Format (TIFF, .tif).

EMF, EPS, PDF, PS, and SVG are vector formats, whereas JPEG, PNG, and TIFF are bitmap formats. If you wish to include a thumbnail of the graph with an EPS file, choose **EPS with TIFF Preview (*.eps)**. Choosing the preview option does not affect how the graph is printed. PNG and SVG are well suited for placing graphs on a webpage. See [G-2] **graph export** for more information.

C.3 Installing Stata for Windows on a network drive

You will need a network license before you can install Stata on a network drive. You can install Stata from the server; or if you have the appropriate privileges, you can install Stata directly to the network drive.

Once Stata is installed, run it to initialize the license. Mount the network drive that Stata is installed on from a workstation. Right-click on the Desktop or on the Windows **Start** menu, and select **New > Shortcut**. Type the path for the Stata executable into the edit field, or click on **Browse...** to locate it. Type Stata as the name of the shortcut.

Once a shortcut for Stata has been created, right-click on it, and select **Properties**. Set the default working directory for Stata by changing the *Start in* field to a local drive that users have write access to. This is where Stata will store datasets, graphs, and other Stata-related files. If the workstation will be used by more than one user, consider changing the *Start in* field to the environment variable %HOMEDRIVE%%HOMEPATH%. Doing so will set the default working directory to each user's home directory.

C.4 Calling Stata from Python

You can call Stata from Python using the pystata Python package. This includes a suite of API functions and IPython magic commands that can be used to interact with Stata and Mata. To learn more about the pystata Python package, view the online documentation at https://www.stata.com/python/pystata. Or see [P] **PyStata module** for more information.

C.5 Changing a Stata for Windows license

If you have already installed Stata and your license needs to be changed, go to **Help > About Stata**, then click on the **Update license...** button on the lower left. You can either browse renewal options or enter a new *License and Activation Key*.

Subject index

This is the subject index for the *Getting Started with Stata for Windows* manual. Readers may also want to consult the combined subject index (and the combined author index) in the *Stata Index*.

A

ado command, 37
arithmetic operators, *see* operators, arithmetic
auto hide, *see* windows, Auto Hide
automatic update checking, *see* updates, automatic
    update checking

B

batch mode, 133–134
blog, *see* Stata Blog
Break button, *see* button, Break
browse command, 2, 7, 62
button,
    Break, 24, 86
    Clear more condition, 24, 135
    Data Browser, 48
    Data Editor (Browse), 2, 24
    Data Editor (Edit), 24, 48, 50
    Do, *see* button, Execute (do)
    Do-file Editor, 24, 98
    Execute (do), 98, 99, 103
    Execute Quietly (run), 103
    Graph, 16, 24
    Graph Editor, 107
    Log, 22, 24, 111
    Open, 24
    Print, 24
    Reset, 18
    Run, *see* button, Execute (do)
    Save, 24
    Show file in Viewer, 104
    Variables Manager, 24
    Viewer, 24, 33
by() option, 14, 17
by prefix, 12, 14, 21
by/if/in tab, 7, 12, 14, 29

C

cd command, 30, 130
clear command, 94
clear more condition, *see* more condition
cmdlog command, 114
codebook command, 5–6, 40
command history, 25
command syntax, 21–22
Command window, 23, 25, 30

commands,
    compared with menus, 21
    downloading and installing, 123–125
    learning, 29, 117–119
    uninstalling, 126
    updating, 126
comments, *see* log files, commenting
contextual menu, 23
    Graph window, 107
    History window, 27
    Variables window, 26
    Viewer window, 36
copy, 48
copying,
    data, *see* data, copying
    graphs, *see* graphs, copying
    text, *see* text, copying
correlate command, 14, 15
creating variables, *see* variables, creating

D

data,
    copying, 53, 67
        tips, 54
    entering, 50–54
    graphing, 15–20
    importing, *see* importing data
    labeling, 72–77
    listing, 78–86
    pasting, 53, 67
    sorting, 55
    strings, 52
Data Editor, 2–3, 48–62
    attach value labels, 56
    browse mode, 62
        toolbar button, 24
    cell navigation, 52
    color coding, 3, 51
    contextual menus, 49
    create value labels, 56
    creating variables, 56
    cursor location box, 50–52, 55
    date
        editing, 59–60
        formats, 59
        mask, 59–60
    edit mode toolbar button, 24
    edit within cell, 52
    empty columns, 51
    empty rows, 51
    enter data by column, 51
    enter data by row, 50
    entering data, 50–54
    filtering, 9, 61
    fonts, 115
    hiding variables, 61

W

Y