10/20/2006

Introduction to STATA 9 for Windows

s

Stata Sizes

Stata is available in several sizes. All have the same features, but differ in the size of data that they can handle:

	Small Stata	Intercooled Stata	Stata SE or MP
Max. number of variables	99	2047	32,766
Max. number of observations	About 1000	Depends on computer's RAM	Depends on computer's RAM

Documentation

Stata Base documentation consists of the following manuals: Getting Started with Stata for Windows; User's Guide; Data Management; 3 volume Reference; and Graphics

Additional documentation is available for Programming, and for various specialized topics including Multivariate Statistics, Survey Data, Survival Analysis, and others. Manuals can be ordered from the Stata website: http://www.stata.com/bookstore/documentation.html

Availability

Stata is available to University of Massachusetts students, faculty and staff for purchase at a discount. Small Stata is only available to students, and only with a 1-year license. Other sizes are available to all, with perpetual licenses. For list of currently available choices and prices, see http://www.stata.com/order/new/edu/gradplan.html

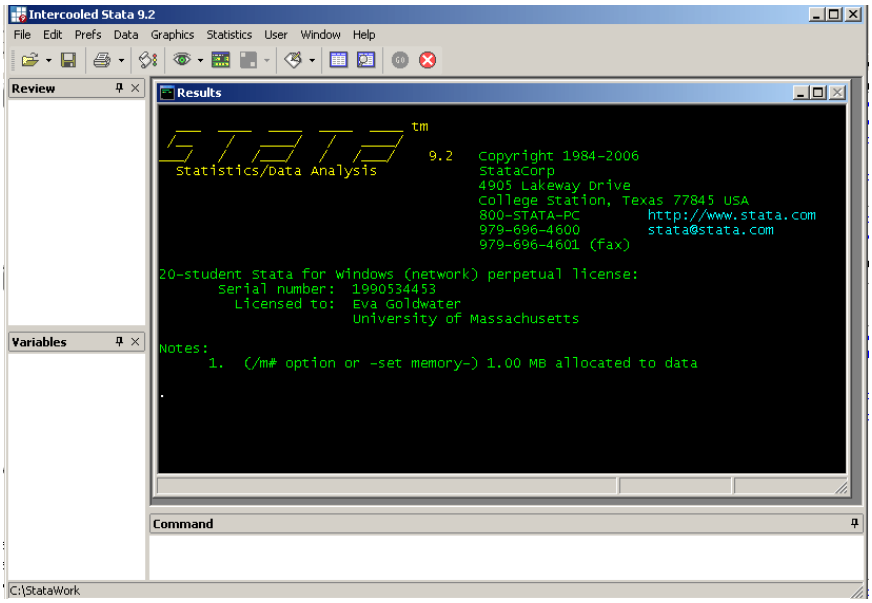
To purchase, call Stata at 1-800-782-8272, or email service@stata.com. Identify yourself as a UMass student, faculty or staff, and make payment directly to Stata. They will fax the University Store to release the software to you.

You can also use Stata in any of OIT's computer classrooms. To find classroom locations and hours, go to http://www.oit.umass.edu/classrooms/about.html

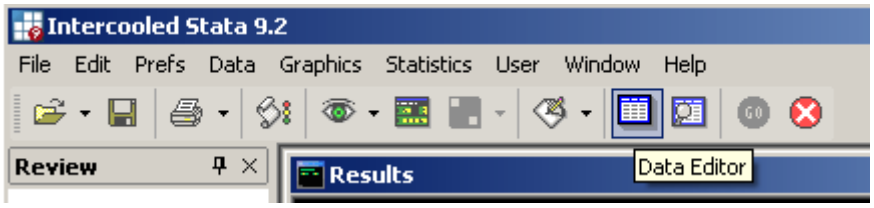
STATA User Interface

When you start Stata, you will see the main Stata interface with four sections:

Command Window: this is where you enter Stata commands
Results Window: displays the results of Stata commands
Review Window: shows the history of Stata commands for this session
Variables Window: displays all the variables in the active data file



The Stata toolbar, located directly underneath the Stata menu, has shortcut icons to the most frequently used commands. To determine what each icon does, hold the mouse pointer over an icon (do not click). A box will appear with a description of the icon function.



For example, holding the mouse over the fourth icon from the right, we see this icon activates the data editor. The data editor is a spreadsheet-like interface used for entering, browsing or editing Stata data.

Stata Language Syntax

Stata is a command driven statistical software package. See "Language Syntax Overview" in the Stata User's Guide for more detail than is provided here.

Many basic Stata commands are available as Menu choices or tool bar icons. When you use the menus, the command is generated for you, and appears in the Results and Review windows, just as if you had typed it. You can use this feature to learn commands, as well as to rerun commands without going through the menus or typing. However not all of Stata's features are available from the menus. To take full advantage of Stata's capabilities, you will need to type Stata commands in the Stata command window. Therefore, this document will focus on Stata commands, not on the menu system.

The general form of a Stata command is:

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

The square brackets denote optional qualifiers. The brackets are not typed when entering a command. **Stata is case-sensititve!** All Stata commands must be lower case.

Since everything except the command is enclosed in square brackets, the simplest form of the language would be to issue just a command. For example, the command:

```
summarize
```

gives the mean, standard deviation and range of all numeric variables in the active Stata dataset.

The meaning of the various parts of the command are:

`by:` prefix requests that the command be executed repeatedly for each distinct value or combination of values of the variable(s) in the list that follows by. For example, if variable `gender` has values M and F, then adding the prefix **`by gender:`** requests that the command be executed twice: once for gender M, and again for gender F. See "Analysis of Subgroups" section for more detail.

`varlist` denotes a list of variable names. Stata variable names can be up to 32 characters long, must start with a letter, and can contain letters, numbers and the underscore (_). Variable names are CASE-SENSITIVE!

The variable list following **by** specifies the by variables (see above); the list following the command specifies the analysis variables for the command.

`command` denotes a Stata command, and is the only part of the general form that is always required. Specific commands may require additional parameters.

`if exp` is an algebraic expression which is used to select the observations to be used by the command. See section "Selecting Cases – if and in" for details.

`in range` is a range of sequential numbers of the observation to be used by the command. See section "Selecting Cases – if and in" for details.

`weight=exp` defines a weighted analysis. Stata supports four kinds of weights, though not all of them are applicable to all commands. See User's Guide section Language Syntax Overview for information about weighting.

`options` is a list of options to be applied to the command. Note that a comma is required to mark the start of the options list. The specific options available vary depending on the command.

Entering and Editing Stata Commands

You type your commands in the Command window. If a command is long, it will wrap onto as many lines as it takes to complete the command. Do NOT press Enter until the very end of the command. When you press Enter, Stata immediately executes the command.

After you press Enter, the command appears in the Review window, followed by the output (if any), or an error message, in the Results window. If you made a typing mistake, click on the command you wish to correct in the Review window. It re-appears in the Command window, ready for you to edit it. Press Enter to submit the corrected command.

If the output from a command does not fit on the Review window, Stata displays as much as fits on the screen, and pauses with the message:

```
--more--
```

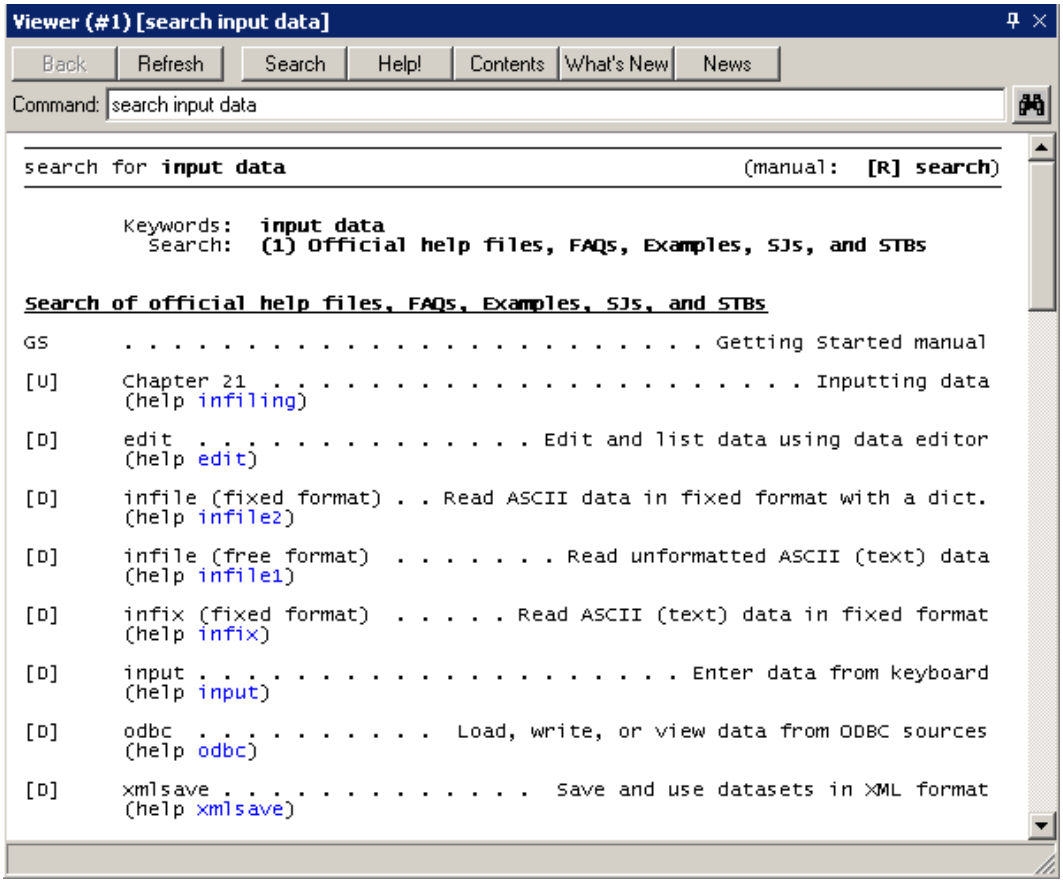
Press the spacebar to display the next screenful of output; press Enter to display just one more line.

The Variables window displays the names of the variables in the file you are using. When you click on the name of a variable in the Variables window, that name is entered in the Command window. Use this feature to help you enter variable lists faster and with fewer errors in the commands you are typing.

Filenames and string values in commands must be enclosed in quotes ("), **not** apostrophes (').

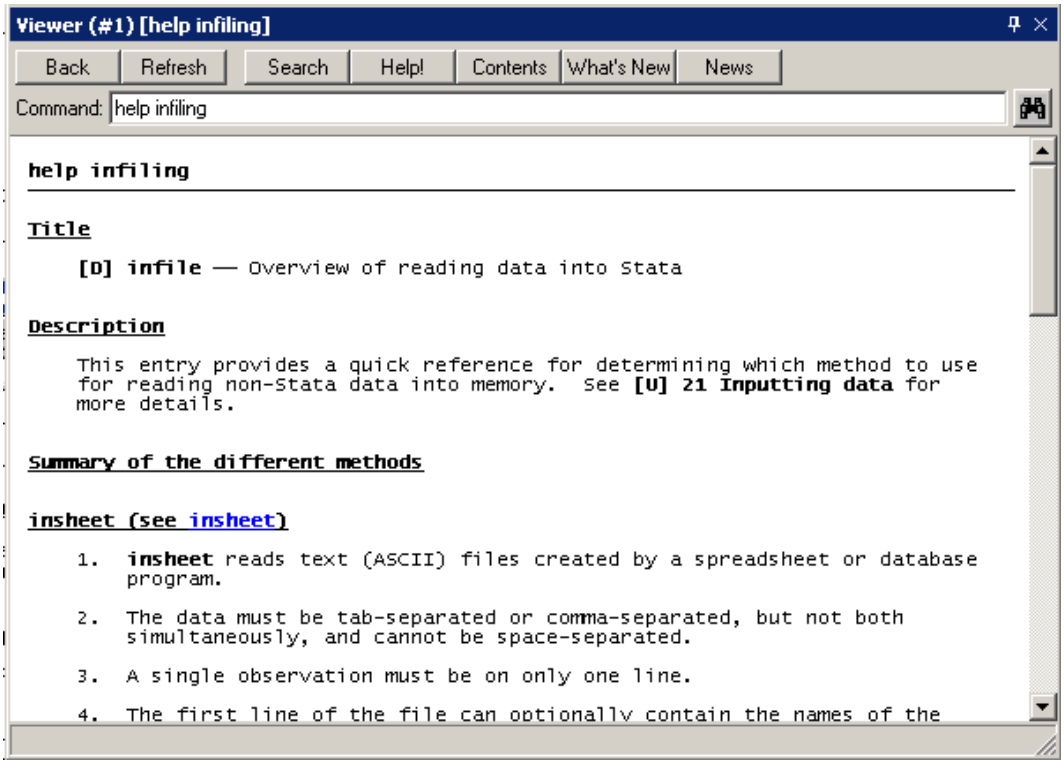
Stata Online Help

Stata has extensive online help. From the Help menu you have the choice of **Contents**, **Search** or **Stata Command** for online help. **Contents** contains a table of contents for all the command help files arranged by subject. **Search** allows you to do a keyword search for help files. For example, if you wanted to find out how to input data you could go to Help → Search, and enter **input data** in the search dialog. You would then get a Help screen similar to the following:

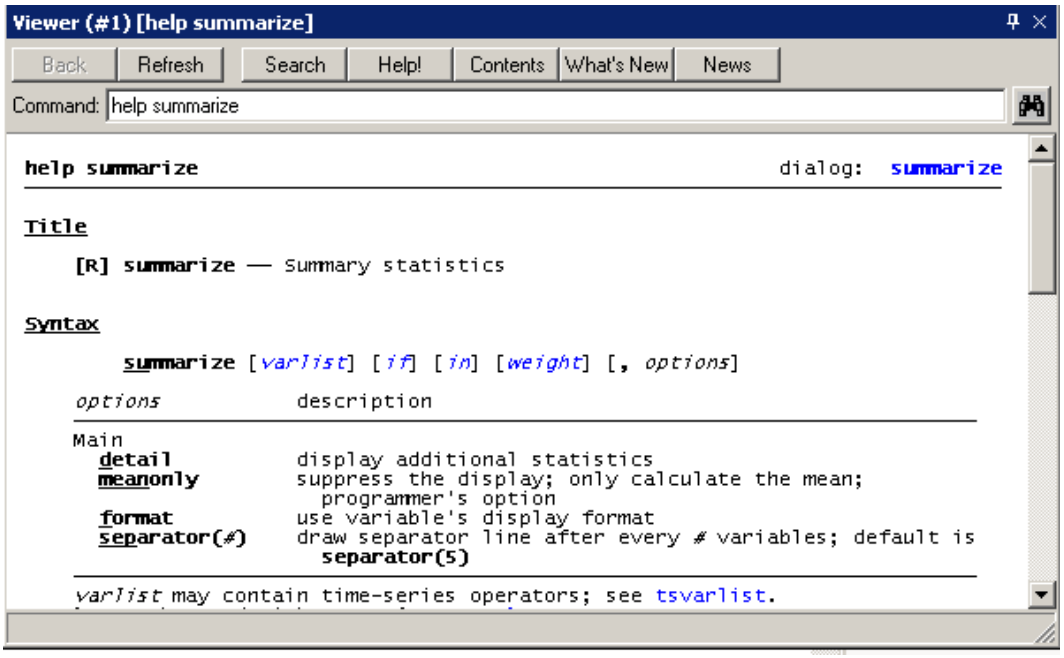


Blue text is a link you can click on for further information. The letter in square brackets on the left indicates the manual that contains information about that command: [U] for Users' Guide, [D] for Data Management, [R] for Reference, [G] for Graphics. Manuals often provide more detail and examples than the online help.

For this example, to get more information about the `infiling` command, we can look in Chapter 21 of the User's Guide, or click on `infiling`. The latter gives the following help screen:



If you know the Stata command that you wish to learn about select **Stata command** from the Help menu and type the command in the dialog box. For example to learn about the summarize command, type summarize in the Stata command dialog to get the following:

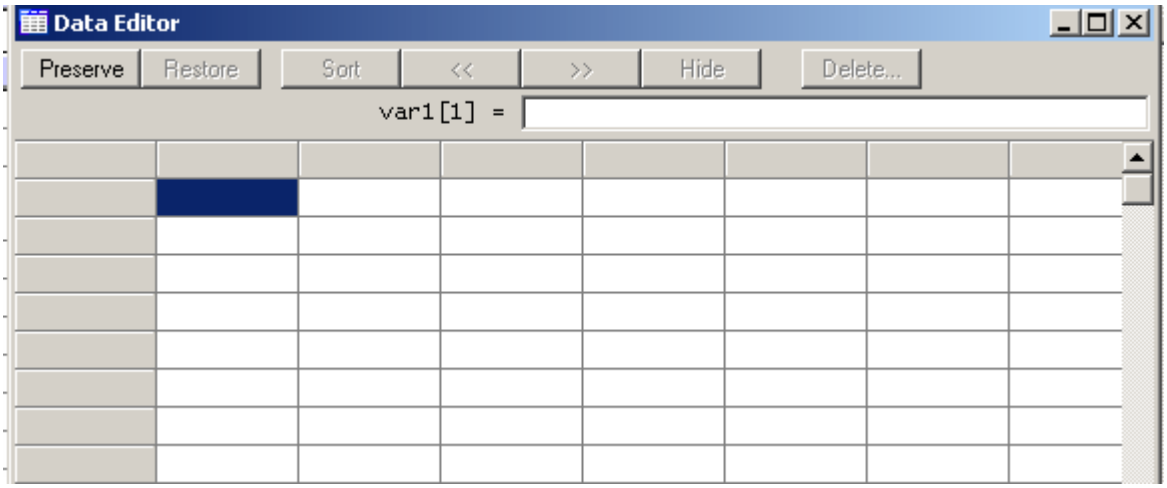


Getting Data into Stata

Stata data consists of observations and variables, corresponding to rows and columns respectively in a spreadsheet-like arrangement. You can only work on one Stata dataset at a time. The data you are working on is stored in memory.

Using the Data Editor

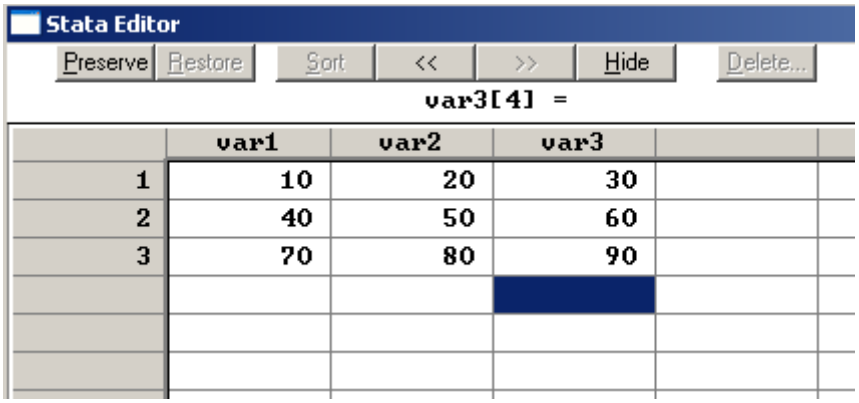
The data editor can be used to enter new data, or to view or edit existing data. To open the Data Editor, select Windows → Data Editor, or click the Data Editor icon (fourth from the right) on the Tool Bar. If you have a dataset in memory, it is displayed in the editor. Otherwise you get a blank editor screen, like this one:



As you type data, it is displayed next to `var1[1]=.` `Var1[1]` stands for variable 1, observation 1, and corresponds to the highlighted cell. When you press Enter, the data you typed is entered into the highlighted cell. The display changes to `var1[2]`, and the corresponding cell is highlighted. Enter the following three observations and three variables:

```
10      20      30
40      50      60
70      80      90
```

The data editor will now look like this:



Stata assigned the default variable names `var1, var2, var3`.

To change the variable names, double-click the column heading `var1`. The Stata Variable Properties dialog opens, with `var1` in the Name field. Change `var1` to your preferred variable name. Here we have changed `var1, var2, var3` to `a,b,c`, respectively. Remember that variable names are case sensitive.

	a	b	c		
1	10	20	30		
2	40	50	60		
3	70	80	90		

You cannot work with data while the Data Editor is open. Click the x in the upper right corner to close the Data Editor. If prompted whether to accept changes, click OK.

Reading Data from Delimited Text Files – "Insheet" Command

The insheet command is used for text files that:

- are comma-delimited or tab-delimited, but *not* space-delimited
- contain 1 observation per line
- may or may not have column headings (If there are no column headings, Stata assigns variable names `v1, v2, v3`, etc. The "rename" command or the Data Editor can later be used to change the names.)

For example, a file arranged like this could be entered into Stata using the insheet command:

```
a,b,c
10,20,30
40,50,60
70,80,90
```

The command to read this file would have the form:

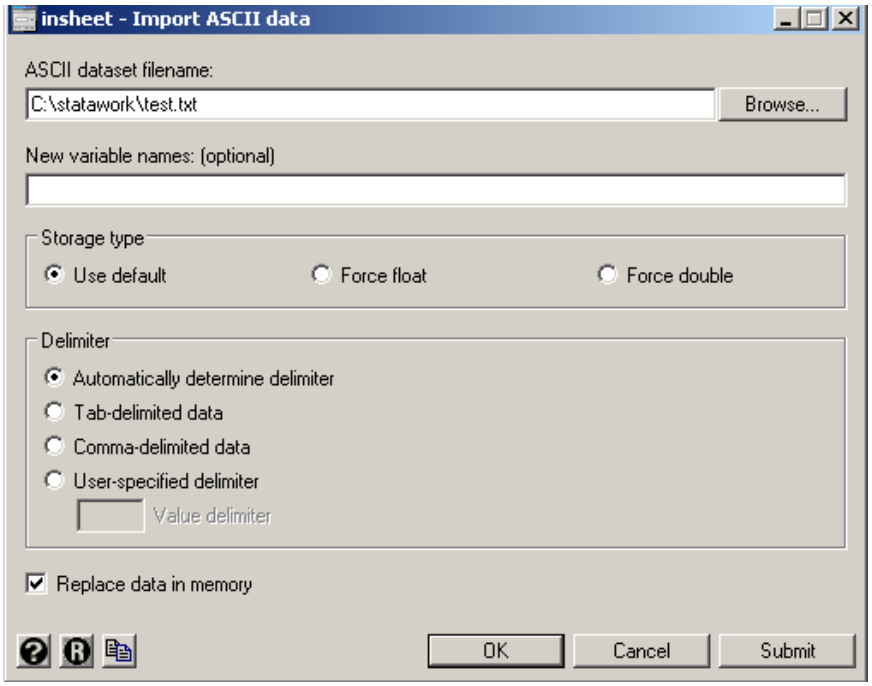
```
insheet using "c:\statawork\test.txt",clear
```

The clear option specifies that any data in memory can be cleared. If you have not saved the data in memory, it is discarded. Because Stata can only have one dataset at a time in memory you need to clear the memory before you can enter new data. Clear can either be used as a stand alone command or as an option on a command that reads new data.

You can generate the `insheet` command from the menus. Select

File → Import → ASCII data created by a spreadsheet

Fill out the `insheet` dialog with the file to be imported; don't forget to check "Replace data in memory". Click OK.



Use the list command, or the Data Editor to see the data.

Reading Data from Excel Files

Excel files, and other spreadsheets, can be read into STATA in one of two ways.

1) Convert the Excel file to .txt (In Excel choose File → Save As and select `.txt` from the "Save as Type" drop down list). Read the text file into STATA with or without column headings using the `insheet` command described above.

2) Copy and paste into Stata. First, in Excel, select and copy your data (with or without column headings). In Stata
   - Type `clear` to clear all data in Stata memory.
   - Open a blank Data Editor window.
   - With the first cell selected, press `ctrl-v` to paste the data into the Data Editor.

Reading Data from Delimited Text Files – "Infile" Command

The `infile` command is used for text files that:

- are comma-, tab-, or space-delimited, or any combination of the three
- contain 1 observation per line, more than 1 observation per line, or 1 observation across multiple lines
- **do not** contain column headings

For the remaining examples we will use data from Appendix A of Minitab Handbook, Second Edition, Ryan, Joiner and Ryan, PWS-KENT Publishing Company, 1985 p. 318. For instructional purposes a date variable was added, a few missing values introduced, and `gender` was coded with letters (M or F) rather than numeric codes. These data are copyrighted and must be acknowledged and used accordingly.

Each subject measured his/her pulse rate. The subjects were then randomly divided into two groups. One group ran in place for two minutes; the other did nothing. At the end of the two minutes, everyone measured his/her pulse again.

The data can be downloaded from:
http://www-unix.oit.umass.edu/~statdata/statdata/data/minidat_date.dat

The description of the data is at:
http://www-unix.oit.umass.edu/~statdata/statdata/data/minidat_date.txt

Download the data file and save it in a convenient location. In this document we will assume that all files are saved in `C:\statawork`.

The data is arranged like this:

Variable	Columns	Description
BDATE	1-10	Date of Birth (mm/dd/yyyy)
PULSE1	12-14	First Pulse (0 = Missing Value)
PULSE2	16-18	Second Pulse (0 = Missing Value)
GROUP	20	Group (1=Ran in place, 2 = Did not run in place)
SMOKE	22	Smokes (1= Yes, 2 = No; 9 = Missing Value)
GENDER	24	Gender (M or F)
HEIGHT	26-28	Height in inches
WEIGHT	30-32	Weight in pounds (0 = Missing Value)
ACTIVITY	34	Usual level of physical activity (1-slight, 2=moderate, 3=very; 9=Missing Value)

Here is a listing of the first few cases:

```
10/15/1985  64  88 1 2 F  66 140 2
03/21/1985  58  70 1 2 M  72 145 2
12/21/1986  62   0 1 1 M  74 160 3
04/05/1986  66  78 1 1 M  73 190 1
07/20/1986   0  80 1 2 M  69 155 2
04/18/1986  74  84 1 2 M  73 165 1
06/12/1985  84  84 1 9 M  72   0 3
08/26/1985  68  72 1 2 M  74 190 2
```

We can read this file into a Stata worksheet using the `infile` command. Type in the Command window:

```
infile str10 bdate  pulse1 pulse2 group smoke  str1 gender height weight  activity
using "C:\statawork\minidat2.dat", clear
```

This is all one command. Since it is long, it will wrap onto two lines as you type. Do not press Enter until the end of the command.

The data file does not have variable names at the top of the file. You provide the variable names (`bdate, pulse1, pulse2`, …) on the `infile` command. Remember that variable names are case-sensitive.

All variables are assumed to be numeric unless you say otherwise,. The "`str10`" preceding variable `bdate` tells Stata to interpret the values of `bdate` as a 10-character string, e.g. "10/15/1985". Similarly `str1` preceding `gender` means that the values of `gender` are 1-character strings ("M" or "F"). If Stata finds non-numeric values in a variable that it expects to be numeric it reports an error similar to this:

```
'03/21/1985' cannot be read as a number for bdate
'M' cannot be read as a number for gender
```

The Results window echoes your command, and reports the result of the command execution. If the data is read successfully you will see:

```
. infile str10 bdate  pulse1 pulse2 group smoke  str1 gender height weight activity
using "C:\statawork\minidat2.dat", clear
(91 observations read)
```

Using the menu system,

File → Import → Unformatted ASCII data

generates the `infile` command. However, the dialog does not provide any means to specify non-numeric variables. Therefore, we cannot use it to read this data file. This is an example of the reason you cannot rely on the menu system if you wish to take full advantage of Stata's capabilities.

Saving a Stata Data File

Stata stores your working data in memory. To save data for future use, you need to save it as a Stata datafile. You can do this using the menus, or the `save` command:

Using the menu, Select File → Save As. In the Save As dialog, browse to the directory where you want to save your file, and enter a filename in the Filename box. Note that Stata data files must have the extension .`dta`.

Use the save command, specifying the location and name for the file to save, in quotes `(")`, not apostrophes `(')`. If you do not specify an extension, Stata will add `.dta` to your filename.

For example:

```
save "c:\statawork\minidat2"
```

If a file with that name already exists, Stata will not save the file unless you add the replace option:

```
save "c:\statawork\minidat2", replace
```

Using a Stata Data File

To use a Stata data file that was saved in a previous session, you use either the menus, or the `use` command:

Using the menu, Select File → Open. Browse to the directory where your file is saved, select the file, and click Open. Since all Stata data files have the extension .`dta`, only those files are available for you to select.

Type the `use` command, specifying the location and name for the file to open, in quotes, not apostrophes. If you do not specify an extension, Stata will add `.dta` to your filename.

For example:

```
use "c:\statawork\minidat2"
```

If there is already some data in memory, Stata will not open the file unless you add the clear option. The clear option tells Stata to discard anything in memory:

```
use "c:\statawork\minidat2", clear
```

Examine the Data

After creating a new data file, you typically need to examine the data to identify possible problems. Two commands for looking at the data directly are `list` and `edit`. Two useful summary commands for this getting brief information about the data are `describe` and `summarize`.

Browse/Edit

The Data Editor lets you view and edit your data. To open the data Editor, click the Data Editor icon on the Stata toolbar – the icon looks like a spreadsheet. The Data Browser displays your data just like the Data Editor, but does not let you make changes. Use it to avoid unintentionally changing your data. Its icon is to the right of the Data Editor icon, and looks like a spreadsheet with a magnifying glass.

You can also open the Data Editor or Browser with the commands:

```
edit
browse
```

You must close the Data Editor or Browser in order to type anything in the Command window, or use the main Stata interface or menu in any way.

When you close the Data Editor, any changes you made are preserved in the copy of the data held in memory, and will be used in all subsequent analyses in this Stata session. If you wish to make the changes permanent for future Stata sessions, you must save the latest data to a file. See **Saving a Stata Data File**.

List

The list command displays your data in the Results window. Like Browse, list does not permit editing the data.

```
list
```

```
Observation 1

       bdate   10/15/1985      pulse1           64      pulse2           88
       group             1      smoke            2      gender            F
      height            66      weight         140    activity            2

Observation 2

       bdate   03/21/1985      pulse1           58      pulse2           70
       group             1       smoke           2      gender            M
      height            72      weight         145    activity            2
Observation 3

       bdate   12/21/1986      pulse1           62      pulse2       0
       group             1       smoke           1      gender       M
      height            74      weight         160    activity       3
--more--
```

--more-- indicates that there is more output to display; Stata paused because the Results window was full. To see the next screen of output, press the space bar.

Describe

The describe command gives you some very basic information about your data – the number of observations (cases), number of variables, and the name and type of each variable. Type in the Command window:

```
describe
```

The Result window displays:

```
  obs:            91
 vars:             9
 size:         3,913 (99.3% of memory free)
-------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------
bdate           str10  %10s
pulse1          float  %9.0g
pulse2          float  %9.0g
group           float  %9.0g
smoke           float  %9.0g
gender          str1   %9s
height          float  %9.0g
weight          float  %9.0g
activity        float  %9.0g
-------------------------------------------------------------
Sorted by:
```

"Storage type" tells you whether a variable is numeric ("float") or character ("str"). In this case, we see that bdate is a 10-character string, and gender is a 1-character string. Everything else is numeric.

"Display format" tells you how Stata will display the values of a variable. %9.0g means the variable will be displayed with up to 9 digits, and Stata will decide whether and how many decimal places to display, depending on context. (The "g" stands for "general".) This works well in most cases, but at times you will need to force Stata to display decimal places. You do this with the `format` command. For example, to force Stata to display weight using 9 digits with 1 decimal place, use the command:

```
format weight %9.1f
```

Formats always begin with the percent sign (%). The "f" stands for "fixed", meaning the number of decimal places to display cannot vary.

We have not assigned any labels, so the label columns are empty.

Summarize

The summarize command displays basic summary statistics for all numeric variables:

```
summarize
```

Here are the Results:

```
    Variable |     Obs        Mean   Std. Dev.       Min        Max
-------------+--------------------------------------------------------
       bdate |       0
      pulse1 |      91    72.64835   13.19459          0        100
      pulse2 |      91     79.8022   19.01766          0        140
       group |      91    1.604396   .4916892          1          2
       smoke |      91    1.769231   .8953823          1          9
      gender |       0
      height |      91    68.73626   3.687321         61         75
      weight |      91    142.2418   27.33916          0        195
    activity |      91    2.208791   .9130715          1          9
```

Notice that bdate and gender have no information listed, not even the number of observations. Character data shows up as missing in many situation. See "String Variables" section for how to deal with this problem.

Also notice that `pulse1`, `pulse2` and `weight` have 0 as the minimum value, and that `smoke` and `activity` have 9 for the maximum value. Looking at the codebook for the data, we see that these are missing value indicators. However, they are included in calculating the means and standard deviations. In order to get the correct calculations, we must tell Stata to exclude these values. This is discussed in the **Missing Values** section.

Missing Values

We noted earlier that missing values coded as numbers are included in all calculations. This is a problem for 0s in variables `pulse1`, `pulse2` and `weight`, and 9s in `smoke` and `activity`. Use the `mvdecode` command to tell Stata to ignore certain values in all calculations.

The following command tells Stata to ignore zero in `pulse1`, `pulse2` and `weight`:

```
mvdecode   pulse1 pulse2  weight, mv(0)
```

The Results window displays:

```
pulse1: 1 missing value generated
pulse2: 1 missing value generated
weight: 1 missing value generated
```

Here is the command to tell Stata to ignore 9s in `smoke` and activity:

```
mvdecode smoke activity, mv(9)
```

The Results window displays:

```
         smoke: 1 missing value generated
      activity: 1 missing value generated
```

To check, let's repeat the summarize command. We see that `pulse1`, `pulse2` and `weight` now have reasonable minimum values, their number of observations is 90 rather than 91, and the mean and standard deviation are different than they were before. Similarly, `smoke` and `activity` no longer include 9.

```
summarize
```

```
    Variable |     Obs        Mean   Std. Dev.       Min        Max
-------------+--------------------------------------------------------
       bdate |       0
      pulse1 |      90    73.45556   10.77467         54        100
      pulse2 |      90    80.68889   17.12849         50        140
       group |      91    1.604396   .4916892          1          2
       smoke |      90    1.688889   .4655417          1          2
      gender |       0
      height |      91    68.73626   3.687321         61         75
      weight |      90    143.8222   22.93399         95        195
    activity |      90    2.133333   .5648904          1          3
```

NOTE: Missing values are stored internally as a very large number. Although they are excluded from all statistical analyses, if you sort the data on a variable that has missing values, the observations with the missing values will be at the end of the file.

Frequencies

Variables `group`, `smoke` and `activity` are categorical, so the above summary with means and standard deviations is not an appropriate way to describe them. In addition, `gender` is also categorical, with non-numeric values. The `tab1` command displays absolute and relative frequencies:

```
tab1 group smoke gender activity
```

This is part of the output, showing the results for `smoke` and `gender`:

```
-> tabulation of smoke

      smoke |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |         28       31.11       31.11
          2 |         62       68.89      100.00
------------+-----------------------------------
      Total |         90      100.00

-> tabulation of gender

     gender |      Freq.     Percent        Cum.
------------+-----------------------------------
          F |         36       39.56       39.56
          M |         55       60.44      100.00
------------+-----------------------------------
      Total |         91      100.00
```

Value Labels

In the tables above, it would be nice to have the values of smoke labeled with their meaning (Yes, No), rather than displayed as 1 and 2. Similarly, the values of group and activity should be labeled for improved readability.

Labeling values has two components – create a label that associates text with the codes, and assign the label to one or more variable.

Use `label define` to create labels. The following creates three labels, names them `yn`, `group` and `activitylevel`, and assigns appropriate text to the numeric codes:

```
label define yn 1 "Yes" 2 "No"
label define group 1 "Run" 2 "No Run"
label define activitylevel 1 "little" 2 "some" 3 "very active"
```

Note that a label may have the same name as a variable, e.g. group. The above labels are not yet associated with any variables. Use `label values` to assign the labels to variables' values:

```
label values group group
label values smoke yn
label values activity activitylevel
```

You can assign the same label definition to more than one variable. If we have several variables with yes/no responses, all coded 1 and 2 respectively, we could assign the yn label to all of them.

Once labels have been assigned, Stata displays these variables using their labels, rather than the numeric codes. For example:

```
tab1 smoke
```

```
      smoke |      Freq.     Percent        Cum.
------------+-----------------------------------
        Yes |         28       31.11       31.11
         No |         62       68.89      100.00
------------+-----------------------------------
      Total |         90      100.00
```

Describe now shows which variables are labeled, and the names of their labels:

```
  obs:            91
 vars:             9                          28 Aug 2006 16:06
 size:         3,913 (99.6% of memory free)
------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
------------------------------------------------------------------------------
bdate           str10  %10s
pulse1          float  %9.0g
pulse2          float  %9.0g
group           float  %9.0g       group
smoke           float  %9.0g       yn
gender          str1   %9s
height          float  %9.0g
weight          float  %9.0g
activity        float  %11.0g      activitylevel
```

If you forget what the codes underlying the labels are, use label list:

```
label list
```

```
activitylevel:
           1 little
           2 some
           3 very active
group:
           1 Run
           2 No Run
yn:
           1 Yes
           2 No
```

If you ever need to see the actual data values, rather than the labels, use the `nolabel` option:

```
tab1 smoke, nolabel
```

```
      smoke |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |         28       31.11       31.11
          2 |         62       68.89      100.00
------------+-----------------------------------
      Total |         90      100.00
```

Tabulations

We've already used the `tab1` command to get frequency counts of the categorical variables `group`, `gender`, `smoke` and `activity`. To get crosstabluation, use the `tab2` command. Here we crosstabulate `gender` by `smoke`:

```
tab2 gender smoke
```

```
-> tabulation of gender by smoke

           |        smoke
    gender |       Yes         No |     Total
-----------+----------------------+----------
         F |         9         27 |        36
         M |        19         35 |        54
-----------+----------------------+----------
     Total |        28         62 |        90
```

Tabulate Options – Percents & Chisquare

Most commands have options to modify how the command operates, or to request more output. Here we use the `row` and `chi2` options on `tab2` to add row percents and a chi-square test of independence to the above table.

```
tab2 gender smoke, row chi2

-> tabulation of gender by smoke

+----------------+
| Key            |
|----------------|
|   frequency    |
| row percentage |
+----------------+

           |         smoke
    gender |       Yes         No |     Total
-----------+----------------------+----------
         F |         9         27 |        36
           |     25.00      75.00 |    100.00
-----------+----------------------+----------
         M |        19         35 |        54
           |     35.19      64.81 |    100.00
-----------+----------------------+----------
     Total |        28         62 |        90
           |     31.11      68.89 |    100.00

          Pearson chi2(1) =   1.0455   Pr = 0.307
```

Summary Statistics

Summarize

We've already used the summarize command to get basic summary statistics for each variable. The detail option adds considerable additional information. Compare the results of summarize without and with the detail option:

```
   summarize weight

    Variable |       Obs        Mean    Std. Dev.       Min
Max
-------------+--------------------------------------------------------
      weight |        90    143.8222    22.93399         95
195


   summarize weight, detail

                          weight
-------------------------------------------------------------
      Percentiles      Smallest
 1%           95             95
 5%          110            102
10%        115.5            108       Obs                  90
25%          125            108       Sum of Wgt.          90

50%          145                      Mean           143.8222
                        Largest       Std. Dev.      22.93399
75%          155            190
90%        177.5            190       Variance        525.968
95%          190            190       Skewness       .2574505
99%          195            195       Kurtosis       2.444199
```

Tabstat

The `tabstat` command provides more flexibility in the choice of summary statistics, and may be more compact than `summarize` with `detail`. You get only the statistics you request:

```
tabstat height weight, stats(n mean sd semean med)
```

```
   stats |    height    weight
---------+----------------------
       N |        91        90
    mean |  68.73626  143.8222
      sd |  3.687321  22.93399
se(mean) |  .3865363  2.417455
     p50 |        69       145
------------------------------
```

Analysis of Subgoups

Most Stata commands can be used with the `by`: prefix to repeat the command for each unique value of one or more categorical variables. Some commands also have a `by()` option, which accomplishes the same thing a little more easily, and usually provides more compact output. The `by:` prefix requires that the data be sorted on the `by` variable(s); the `by()` option has no such requirement.

Let's repeat the above summary statistics for height and weight, but this time we'd like to see the results separately for males and females. For brevity, we'll only ask for the mean.

by() option

The `tabstat` command has a `by()` option, so we'll try that first:

```
tabstat height weight, stats(mean) by(gender)

Summary statistics: mean
  by categories of: gender

gender |    height    weight
-------+--------------------
     F |  65.41667  123.6944
     M |  70.90909  157.2407
-------+--------------------
 Total |  68.73626  143.8222
---------------------------
```

by: prefix

Now, let's try it with the `by:` prefix:

```
by gender:tabstat height weight, stats(mean)
not sorted
r(5);
```

We get an error, because the data is not sorted by gender. We try again, this time sorting the data first:

```
sort gender
by gender:tabstat height weight, stats(mean)

-------------------------------------------------------
-> gender = F

   stats |    height    weight
---------+--------------------
    mean |  65.41667  123.6944
-----------------------------

-------------------------------------------------------
-> gender = M

   stats |    height    weight
---------+--------------------
    mean |  70.90909  157.2407
-----------------------------
```

Additional Analyses

Confidence Intervals

To get 95% confidence intervals for normally distributed variables, use the `ci` command:

```
ci height weight
```

```
 Variable |        Obs        Mean    Std. Err.       [95% Conf. Interval]
 ---------+--------------------------------------------------------------
   height |         91    68.73626    .3865363        67.96834    69.50419
   weight |         90    143.8222    2.417455        139.0188    148.6257
```

Add the `level` option to get different confidence intervals:

```
ci height weight, level(90)
```

```
 Variable |        Obs        Mean    Std. Err.       [90% Conf. Interval]
 ---------+--------------------------------------------------------------
   height |         91    68.73626    .3865363        68.09386    69.37867
   weight |         90    143.8222    2.417455         139.804    147.8404
```

Two sample t-test

We use a two sample t-test to see whether runners and non-runners differ in pulse rate after running:

```
ttest pulse2, by(group)
```

```
Two-sample t test with equal variances
------------------------------------------------------------------------------
   Group |     Obs        Mean    Std. Err.   Std. Dev.   [95% Conf. Interval]
---------+--------------------------------------------------------------------
     Run |      35        93.2    3.171247    18.76135    86.75525    99.64475
  No Run |      55    72.72727    1.320508    9.793147    70.07981    75.37473
---------+--------------------------------------------------------------------
combined |      90    80.68889    1.805502    17.12849     77.1014    84.27638
---------+--------------------------------------------------------------------
    diff |            20.47273    3.018231                14.47463    26.47083
------------------------------------------------------------------------------
    diff = mean(Run) - mean(No Run)                               t =   6.7830
Ho: diff = 0                                     degrees of freedom =       88

    Ha: diff < 0                 Ha: diff != 0                 Ha: diff > 0
 Pr(T < t) = 1.0000         Pr(|T| > |t|) = 0.0000          Pr(T > t) = 0.0000
```

Paired t-test

Now let's compare pulse before and after running. To do this, we need to use a paired t-test. Further, we'll use the `by` prefix to get separate analyses for the runners and non-runners. To use the `by` prefix, the data must be sorted by `group`:

```
sort group
by group:ttest pulse1=pulse2
```

```
-> group = Run

Paired t test
------------------------------------------------------------------------------
Variable |     Obs        Mean    Std. Err.   Std. Dev.   [95% Conf. Interval]
---------+--------------------------------------------------------------------
  pulse1 |      34    74.61765    1.939373    11.30839    70.67196    78.56333
  pulse2 |      34    93.58824    3.241365    18.90024    86.99363    100.1828
---------+--------------------------------------------------------------------
    diff |      34   -18.97059    2.680718    15.63114   -24.42455   -13.51663
------------------------------------------------------------------------------
     mean(diff) = mean(pulse1 - pulse2)                           t =  -7.0767
 Ho: mean(diff) = 0                        degrees of freedom =       33

 Ha: mean(diff) < 0           Ha: mean(diff) != 0           Ha: mean(diff) > 0
 Pr(T < t) = 0.0000         Pr(|T| > |t|) = 0.0000          Pr(T > t) = 1.0000
```

The above is the output for the first group. This is followed by the output for the second group, which we do not show here.

Selecting Cases

The `if` and the `in` parameters can be added to any Stata command to select a subset of cases to be used.

if

`if` selects cases that satisfy some logical criterion; for example, we can restrict the above paired t-test to the first group:

```
ttest pulse1=pulse2 if group==1
```

Notice the double equal sign in the `if` portion of the command. Stata uses the double equal sign in all logical conditions. `group==1` is a logical condition because we are asking Stata to check whether the value of variable group is 1, and only include it in the analysis if it is. The single equal sign is reserved for assigning values to variables – see "Creating New Variables".

Note that although group is labeled, and displayed as "Run" and "No Run", we must use its actual numerical value in the `if` condition.

You can combine several conditions on the `if` criterion. Here, we request the analysis to include only male runners:

```
ttest pulse1=pulse2 if group==1 & gender=="M"
```

The symbols to use for building logical expressions are:

```
      Logical            (numeric and string)
------------------     ---------------------
    ~   not              >   greater than
    !   not              <   less than
    |   or               >=  > or equal
    &   and              <=  < or equal
                         ==  equal
                         ~=  not equal
                         !=  not equal
```

in

`in` selects cases based on their position in the datafile. For example, to list the first 3 observations:

```
list in 1/3
```

To list observations from the end of the file, use negative numbers. This lists the last 3 observations:

```
list in -3/-1
```

Creating New Variables

You can create new variables using formulae and existing variables. Two useful commands for doing this are `generate` and `recode`.

generate and replace

`generate` (abbreviated `gen`) is used to calculate new variables using arithmetic, algebraic and/or logical constructs. In order to examine the change in pulse, let's compute the difference in pulse rates between the second and first measurements for each student. We'll call the new variable "pdiff". We use `generate` to calculate the new variable:

```
gen pdiff=pulse2-pulse1
```

Notice that the Result window says:

```
(2 missing values generated)
```

Variables `pulse1` and `pulse2` each had one missing value. The difference cannot be computed when either of the operands is missing. Therefore, the newly computed difference variable has 2 missing values.

If you make a mistake in the generate command, and try to re-do it with a correction, you get an error:

```
gen pdiff=pulse2 – pulse1
pdiff already defined
```

In order to recalculate an existing variable, use replace instead of generate:

```
replace pdiff=pulse2 – pulse1
```

Basic arithmetic expressions are formed using the operators:

```
+   addition  (numeric) or concatenation (string)
-   subtraction
*   multiplication
/   division
^   power
```

recode

`recode` is used to make or redefine categories. The following command divides variable `height` into 5 categories. We use the `gen` option on recode to store the result in a new variable called `ht_cat`. Without this, the original values of `height` would be lost.

```
recode height (min/65=1)(66/68=2)(69/72=3)(73/max=4), gen(ht_cat)
```

To see the results, we run `tab1` on the new variable:

```
tab1  ht_cat

  RECODE of |
     height |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |         17       18.68       18.68
          2 |         27       29.67       48.35
          3 |         30       32.97       81.32
          4 |         17       18.68      100.00
------------+-----------------------------------
      Total |         91      100.00
```

It would be good to label the newly created categories. We could do this by defining value labels and associating them with variable `ht_cat`, as shown under "Value Labels". Alternatively, `recode` allows you to create labels as you recode. (Note: the command below will wrap to the next line; do not press Enter until the end of the command.)

```
recode  height (min/65=1 "65 or less") (66/68=2 "66-68")(69/72=3 "69-72")
 (73/max=5 "73 or more"),  gen(ht_cat) label(ht_cat_label)
```

The tabulation now displays the label for each category.

```
tab1 ht_cat

-> tabulation of ht_cat

  RECODE of |
     height |      Freq.     Percent        Cum.
------------+-----------------------------------
 65 or less |         17       18.68       18.68
      66-68 |         27       29.67       48.35
      69-72 |         30       32.97       81.32
 73 or more |         17       18.68      100.00
------------+-----------------------------------
      Total |         91      100.00
```

Graphs

Stata has extensive graphical capabilities, with many options. We will introduce only a few simple types of graphs and options. For (lots) more information, consult the Stata Graphics manual.

All graph commands begin with:

```
graph graphtype
```

Where *graphtype* specifies the kind of graph. The most common `graphtypes` are `bar`, `box`, `matrix`, `pie`, and `twoway`. Twoway graphs require additional specification of the kind of `twoway` graph: `area`, `bar`, `connected`, `histogram`, `line`, and `scatter` are some of the choices within the `twoway` family. These lists are by no means complete.

Many options are available for adding titles, labeling axes, and controlling other features. The options available will vary depending on the type of graph.

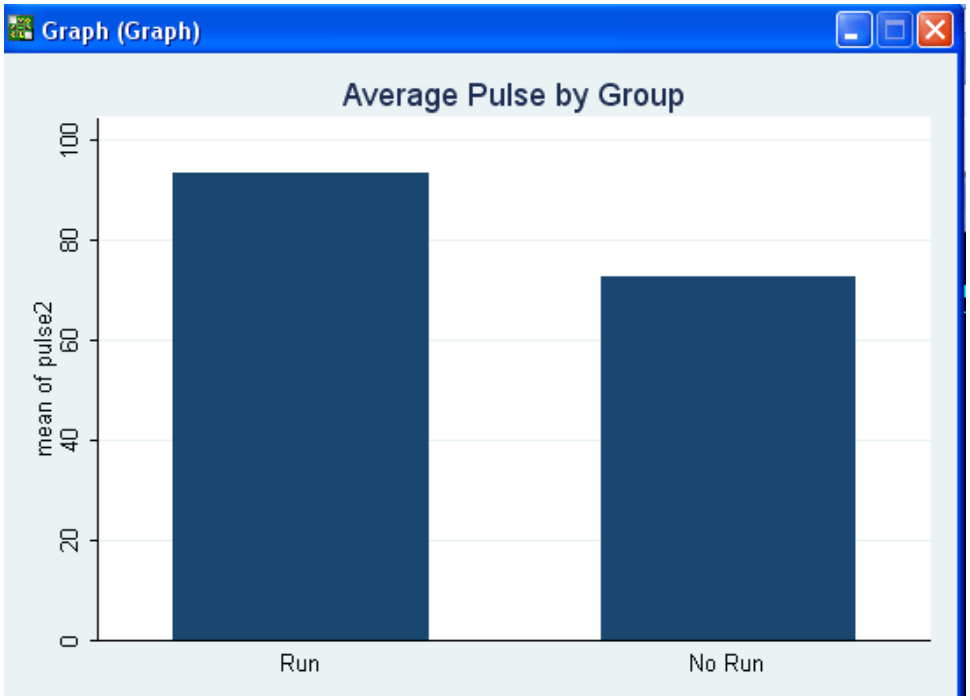
Descriptive Statistics Bars

`graph bar` is used to make bar charts displaying descriptive statistics (such as mean, median, percentiles, sum and others). Typically, such a graph is used to visually compare groups. For example, to compare average `pulse2` for the two groups (runners and non-runners):

```
graph bar (mean) pulse2, over(group)
```

To get a title, add:

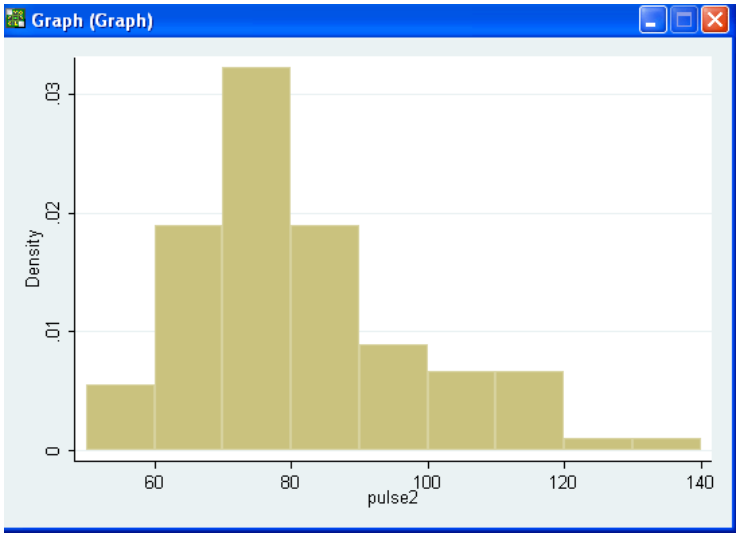
```
graph bar (mean) pulse2, over(group) title("Average Pulse by Group")
```



Histograms

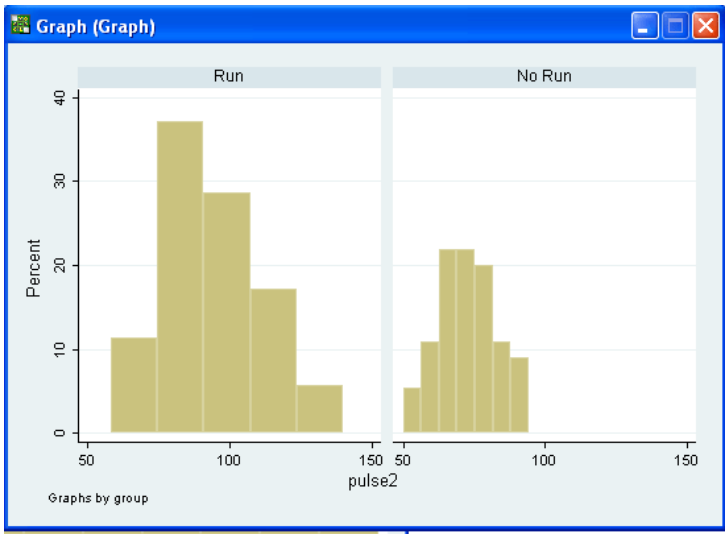
Histograms show the distribution of continuous variables. Stata chooses appropriate cutpoints for the bars:

```
graph twoway histogram pulse2
```



We would expect the distribution of `pulse2` to be quite different for runners and non-runner. We add the `by()` option to get separate histograms for the two groups. In addition, we'd like to see the histogram scaled in percent, rather than density.

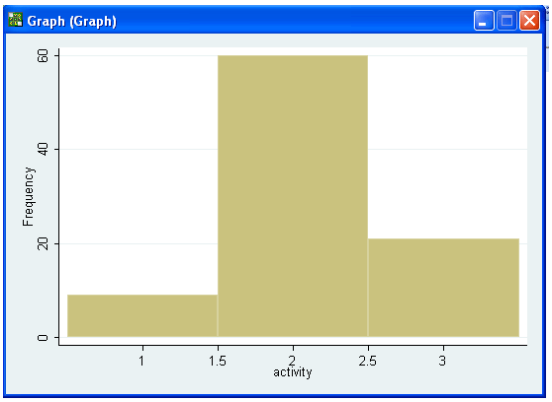
```
graph twoway histogram pulse2, by(group) percent
```



Frequency Bars

Histogram graphs can also be used to display frequency distributions for a categorical variable. To do this, just add the `discrete` option to the histogram. Here we also use the `freq` option to display the number of subjects (rather than density) in each of the three activity levels:

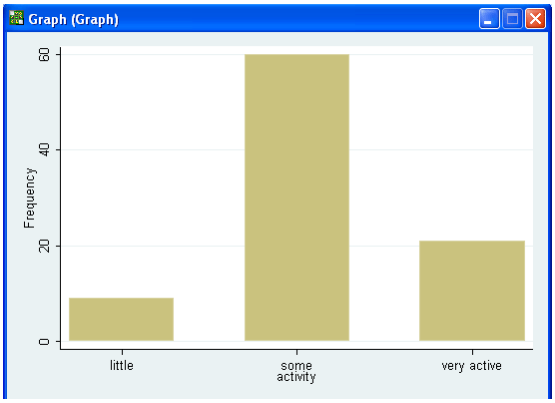
```
graph twoway histogram activity, discrete freq
```



This graph would look nicer if the bars were separated by some blank space. Also, the scale for activity should only show the integers, 1, 2, and 3, and these should be labeled with their assigned labels, rather than with numbers.

To put space between the bars, we add the `gap(#)` option, where # is the percent of the bars' width to leave blank between bars. The `xlabel(1(1)3 valuelabel)` option specifies that the x-axis (horizontal) should be labeled starting at 1, with increments of 1, ending at 3, and displayed with value labels.

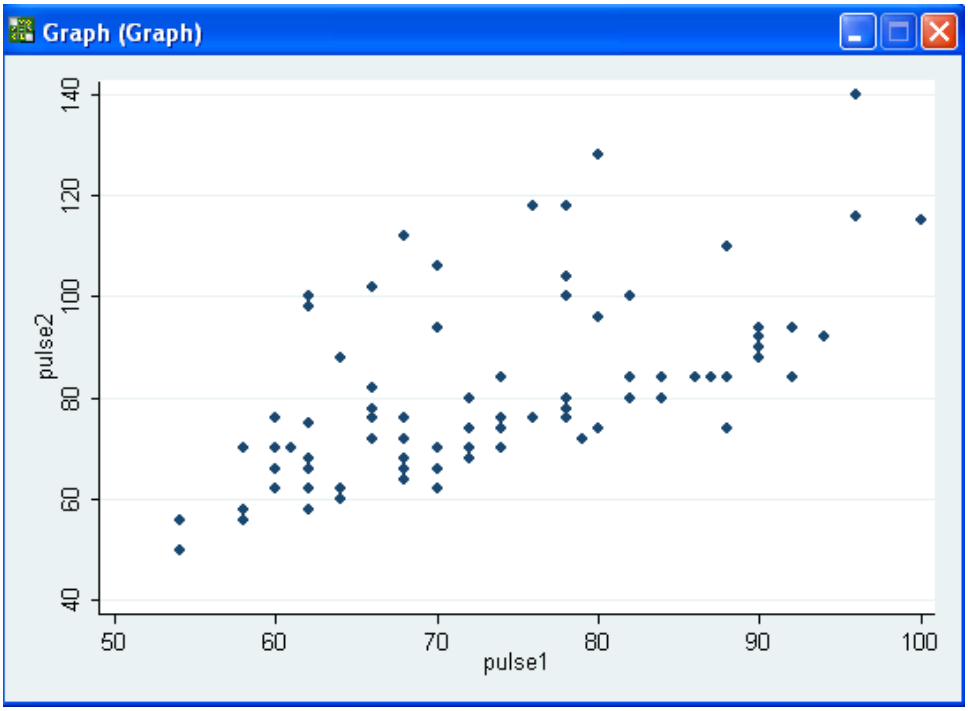
```
graph twoway histogram activity, discrete freq gap(40) xlabel(1(1)3,
valuelabel)
```



Scatterplots

Here is a simple scatterplot of `pulse2` versus `pulse1`:

```
graph twoway scatter pulse2 pulse1
```

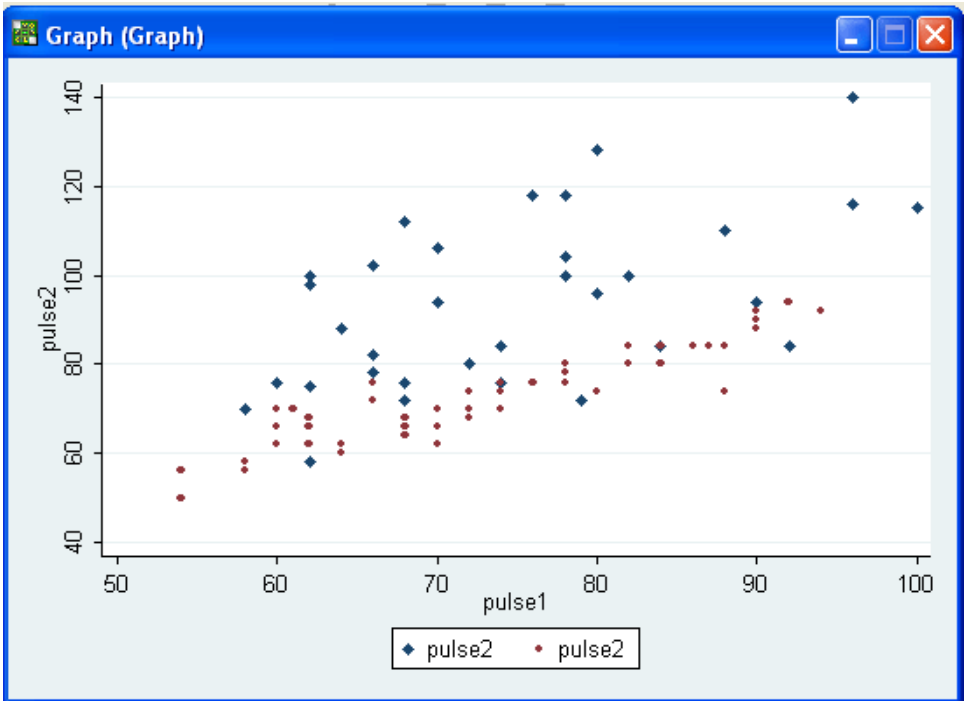


Overlay Graphs, Symbols & Legends

The above scatterplot does not distinguish the runners from the non-runners. You could add the `by(group)` option to get side by side graphs of the two groups. But what if you want to see a single graph, with distinct markers to distinguish the groups? For this you need to request two graphs on one set of axes.

Use parentheses to specify graphs to be overlaid. The first graph is a scatterplot of `pulse2` versus group 1 (runners), the second of `pulse2` versus group 2 (non-runners). The `msymbol` option specifies small diamonds for the first group, and small circles for the second. (This is one long command, which wraps onto 2 lines.)

```
graph twoway (scatter pulse2 pulse1 if group==1, msymbol(d))
(scatter pulse2 pulse1 if group==2, msymbol(o))
```

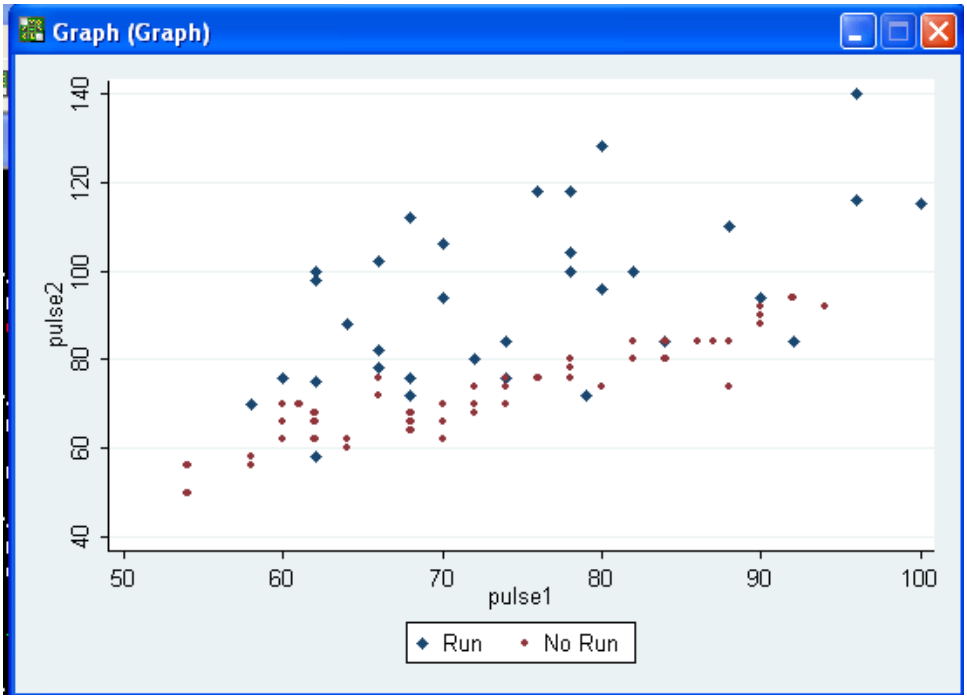


There's one problem. The legend labels the symbols with the name of the y-variable, rather than with the name of the group that the symbol represents. We add the legend option to label the symbols with the group names:

```
graph twoway (scatter pulse2 pulse1 if group==1,msymbol(d))
   (scatter pulse2 pulse1 if group==2, msymbol(o)),
    legend(order(1 "Run" 2 "No Run"))
```

Notice the arrangement of parentheses and options – each scatter command is enclosed in parentheses, and has its own option (msymbol) following a comma.

These options apply only to the preceding scatter, inside the parenthesis. The legend option follows the comma after the closing parenthesis of the last scatter. This option applies to the entire graph.



String Variables

Recall that variables `gender` and `bdate` are string:

```
describe gender bdate

              storage  display     value
variable name   type   format      label      variable label
------------------------------------------------------------------------------
gender          str1   %9s
bdate           str10  %10s
```

`gender` has values "M" and "F". Stata severely limits what you can do with string (non-numeric) variables. You can use them in `tabulate`, or as `by` variables, but for most other purposes you need numeric data. For example, in the previous section we made a bar chart of the frequencies of each value of `activity`, using the command:

```
graph twoway histogram activity, discrete freq
```

We should be able to make a bar chart of the frequencies of each value of `gender`, using the similar command:

```
graph twoway histogram gender, discrete freq
```

Instead of the expected chart we get an error:

```
varlist:  gender:  string variable not allowed
r(109);
```

String variables must be converted to numeric form in order to use them in most commands, such as the frequency bar chart. The `encode` command is a convenient way to create a new variable with numeric codes corresponding to the distinct values of a string variable:

```
encode gender, gen(sex)
```

This generates a new numeric variable, `sex`, and associated labels based on the codes found in `gender`. The values of `sex` are integers, assigned alphabetically. Thus `sex` is assigned value 1 with label "F", and 2 with label "M". `describe` shows that `gender` is string, while `sex` is numeric, with assigned label `sex`.

```
describe gender sex

              storage  display     value
variable name   type   format      label      variable label
------------------------------------------------------------------------------
gender          str1   %9s
sex             long   %8.0g       sex
```

To see how the labels were assigned, use `label list`. Here is the relevant snippet of output:

```
sex:
           1 F
           2 M
```

You can confirm that `sex` and `gender` have the same values and frequencies:

```
tab1 gender sex

-> tabulation of gender

     gender |      Freq.     Percent        Cum.
------------+-----------------------------------
          F |         36       39.56       39.56
          M |         55       60.44      100.00
------------+-----------------------------------
      Total |         91      100.00

-> tabulation of sex

        sex |      Freq.     Percent        Cum.
------------+-----------------------------------
          F |         36       39.56       39.56
          M |         55       60.44      100.00
------------+-----------------------------------
      Total |         91      100.00
```

Now you can use the numeric variable `sex` to get the bar chart of frequencies. We also add a gap between the bars, and label the integer values, using the assigned labels:

```
graph twoway histogram sex, discrete freq gap(60) xlabel(1 2,
valuelabel)
```

Other commands useful for changing from string to numeric values are `destring` and `real`. Both of these are used for converting string variables that contain values that *look* like numbers, to actual numbers.

Dates

The variable "bdate" was read as a string date. If you sort the data by bdate, you will get "01/11/1987" before "01/19/1986".

```
sort bdate
list bdate
```

```
     +------------+
     |      bdate |
     |------------|
  1. | 01/06/1987 |
  2. | 01/09/1987 |
  3. | 01/11/1987 |
  4. | 01/19/1986 |
  5. | 01/20/1987 |
     |------------|
  6. | 01/21/1985
```

Further, we cannot do any calculations, such as computing age, using bdate. In order to do anything reasonable with bdate, we need to tell Stata to interpret it as a date. When we tell Stata to interpret a string as a date, it stores the information as the number of days since January 1, 1960. Here we create a new variable, birthdate, which is a Stata date, and sort the data according to birthdate:

```
generate birthdate=date(bdate, "mdy")
format birthdate %d
sort birthdate
list birthdate
```

```
     +-----------+
     | birthdate |
     |-----------|
  1. | 06dec1983 |
  2. | 09feb1984 |
  3. | 18feb1984 |
  4. | 17apr1984 |
  5. | 20apr1984 |
     |-----------|
  6. | 05jun1984 |
```

generate creates the new variable, birthdate, to store the result of the date function.

The date function is used to interpret strings that ***look*** like dates into Stata dates. The "mdy" in the date function specifies the order in which month day and year appear in the string. (If year has only 2 digits you'll need an additional parameter to interpret the string correctly – see Help → Search → date function.)

Finally, the `format` command formats the new variable as a date. Without it, birthdate would print as number of days since January 1, 1960.

We can now drop the original string variable, which serves no further useful purpose:

```
drop bdate
```

Date variables can be used in calculations and in graphs. For example, here we compute each person's age as of January 1, 2006. Since Stata dates are measured in days, we divide by 365.25 to get age in years:

```
generate today=date("1/1/2006","mdy")
generate age=(today-birthdate)/365.25
```

Efficiency Tricks

Using the Review and Variables Windows

The **Review** window displays your past commands. If you click on a command in the Review window it is copied to the Command window, so you can edit and run it again. If you double-click on a command in Review, it is immediately executed.

The **Variables** window lists the variables in the Stata dataset. As you type commands, you can click on a variable in the Variable window to copy it to the Command window at the cursor position.

Both commands and variable names can be abbreviated to the shortest string that uniquely identifies them. For example, earlier we ran the command:

```
summarize weight
```

This could have been abbreviated (though for the sake of clarity, such extreme abbreviations are not advisable):

```
sum w
```

For variables, it is much better to use the Variable window to quickly generate the full name.

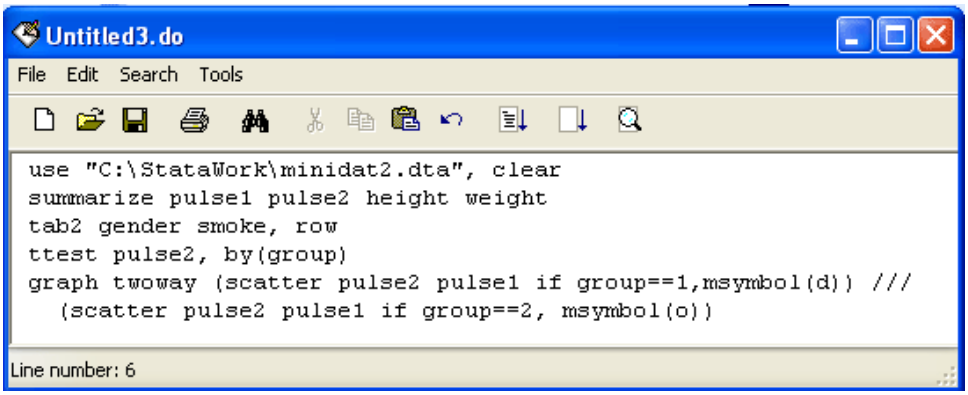
Using the Do-File Editor:

The do-file editor enables you to type a series of Stata commands and submit them all at once. You can save the commands in your do-file, so you can later reproduce, edit or add to your work without having to re-type commands.

To open the do-file editor, click on the Do-file Editor icon (it looks like an envelope) or select Window → Do-file Editor → New do-file.

The do-file editor has basic features of any text editor: cut, copy, paste, undo, open, save and print. Here is the Do-file editor with commands to

- open the `minidat2` Stata data file
- run summary statistics on `pulse1`, `pulse2`, `height` and `weight`
- tabulate `gender` by `smoke` with row percents
- compare `pulse2` between the run and no-run groups using a t-test
- make a scatterplot of `pulse1` against `pulse2` with the run and no-run groups indicated by different markers.



Note the three slashes (///) in the graph command. This is a continuation symbol you can use (in the Do-editor only, not on the Stata command line) if you need to break a long command on to more than one line.

To execute all the commands, select Tools →Do

If you want to execute only some of the commands in the Do-file editor, select the commands you want, and use Tools → Do Selection.

You can of course save the contents of the Do-file editor, and Open them to use again in a future session. Note that the Save and Open file menu selections in the Do-file editor window can only be used to save and open do files; the Save and Open file menu selections in the Stata main window only save and open data files.

Saving the Review Window as a Do-File:

Every command you type in the Command window goes to the Review window. In addition to retrieving commands from the Review window for immediate re-execution or editing, you can save the contents of the Review window as a do-file. To do this, Right-click anywhere in the Review window (not on its title bar) and select Save Review Contents.

Using Log to Print and Save Output:

To print the contents of the Results window, select

File → Print →Results.

This prints everything in the Results window. You cannot select parts to print.

If you want to edit or print parts of the Results, you must capture the output in a log file. To start a log file select

File → Log → Begin

or click on the Begin Log button (if looks like a scroll). When the dialog box appears, fill in a name and select the type of log file (*.smcl or *.log).

- *.scml is a Stata formatted log; it can be printed from Stata, in whole or part, retaining bolds, underlines, and italics as you see them in the Results window. *.scml logs cannot be edited.

- *.log is a plain text file; it can be opened for editing or printing in any text editor or word processor.

When you begin a log file, all *subsequent* text output goes to both the Results window to the log file. Graphs do not go to the log. To stop collecting output in the log, select

File → Log → Suspend (or Close).

To print output from the log, open the log in Stata's Viewer. Select

File →Log →View

and browse to your log file. Select the portion you want to print. From Stata's main menu, select

File → Print Viewer

If a portion of the log is highlighted, you can print just that part of the log by choosing "Selection" on the print dialog.

You can type a title into the "Header" field, which will print at the top of each page. Uncheck the "Print Logo" box.

The log file displayed in the Viewer is a snapshot taken at the time the Viewer is opened. To update the log in the Viewer, click on the Refresh button at the top of the Viewer window.