2018–2024 Tue Gørgens

Introduction to Stata #1

1 Introduction

Self-study In the first week, we will spend the tutorial time on getting familiar with Stata. There will be more Stata material in the coming weeks, more than can be covered during the tutorial sessions. However, this series of introductions to Stata is suitable for self-study, so make sure you complete it on your own. *If you have difficulties or questions, please do not hesitate to consult your tutor!* If you have suggestions for how to improve the tutorial, please let me know.

Why Stata In this course you will learn how to use the computer application Stata for data analysis. There are several different applications available that can perform econometric and statistical data analyses, each with their specific strengths. Traditionally, Stata's strengths were in the analysis of cross-section data, but over the years a full range of tools for time series analysis has been added, so you can now do everything in Stata.

What is ahead This document will get you started working with Stata. There will be more parts coming over the next several weeks, to strengthen and expand your Stata knowledge and skills. (Note this introduction is written for version 15. New features have been added in later versions of Stata, but we will not use them in this course.) Generally, the material is organised to go from elementary to advanced as the course progresses, so many topics will revisited several times. The main focus at this stage is on how to manage data and work effectively with Stata. Specific tools for econometric analysis will be introduced as the course progresses.

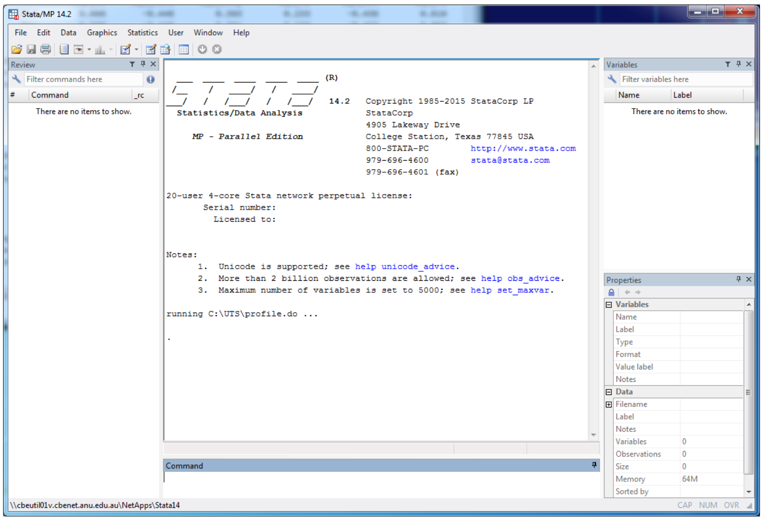
In part #1, we begin with the Stata interface where you tell Stata what you want and Stata gives you what you asked for. (Which may or may not be the same of course.) Then we consider the Stata dataset, different ways to look at its content, and how to create new variables. Then we go through the various kinds of formulae, 'expressions', that Stata understands. Finally, we perform simple econometric/statistical analyses such as computing descriptive statistics (means, variances), constructing frequency tables, and creating histograms.

Part #1 uses the dataset `NHISmodif.dta` in many of the examples. You can download this file from Wattle.

Is it 'dataset' or 'data set'? The word 'dataset' is often used in academia, but it doesn't appear in any dictionary I know of. The Stata documentation and online help uses 'dataset', and we'll follow their style here.

2 Interacting with Stata: the interface

The main window The Stata interface is highly customisable, so may look a bit different depending on the settings of the last user. Basically, there are five sub-windows within the main window: Command, Review, Variables, Properties, and an untitled window in the middle called Results. When you initially start Stata, perhaps they are arranged like this:



```
  ___  ____  ____  ____  ____ (R)
 /__    /   ____/   /   ____/
___/   /   /___/   /   /___/   14.2   Copyright 1985-2015 StataCorp LP
  Statistics/Data Analysis            StataCorp
                                      4905 Lakeway Drive
     MP - Parallel Edition            College Station, Texas 77845 USA
                                      800-STATA-PC        http://www.stata.com
                                      979-696-4600        stata@stata.com
                                      979-696-4601 (fax)

20-user 4-core Stata network perpetual license:
       Serial number:
         Licensed to:


Notes:
      1.  Unicode is supported; see help unicode_advice.
      2.  More than 2 billion observations are allowed; see help obs_advice.
      3.  Maximum number of variables is set to 5000; see help set_maxvar.

running C:\UTS\profile.do ...

.
```

Variables
Name
Label
Type
Format
Value label
Notes
Data
Filename
Label
Notes
Variables 0
Observations 0
Size 0
Memory 64M
Sorted by

The five basic windows serve different purposes:

Command The Command window is for giving instructions, or ‘commands’, to Stata. For example, you might want to fit a model to some data and you type the specification here (in a syntax that Stata understands), or you might want Stata to create a graph for you. Hit the Enter key when you have finished typing and are ready to roll.

Results Stata will repeat your commands in the Results window, followed by the results or perhaps other messages, errors, warnings, etc depending on the particular instruction.

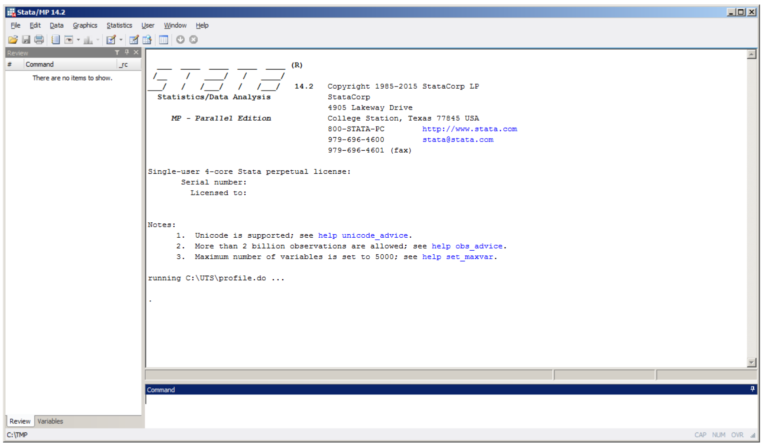
Review The Review windows shows a list of all your previous instructions (in the current session). If you want to repeat an instruction you can click on it to copy it back to the Command window.

Variables Stata always works on one dataset at a time, and the names of the variables in the currently loaded dataset are listed in the Variables window. You can also copy variable names to the Command window by clicking on them.

Properties The Properties window shows additional information about the variables and the dataset itself.

Stata has other windows that typically pop up outside the main window when called for. For example, there are Graph, Viewer, Data Editor, and Do file Editor windows. More on these later.

You can resize, rearrange and close windows as you like, and Stata normally remembers this next time you begin. For example, on my own computer I prefer to have a larger Results window, so I have closed the Properties window and moved the Variables behind the Review window:


Stata/MP 14.2
File Edit Data Graphics Statistics User Window Help
Review
# Command _rc
There are no items to show.
Command
Review Variables
C:\TMP
CAP NUM OVR

```
  ___  ____  ____  ____  ____ (R)
 /__    /   ____/   /   ____/
___/   /   /___/   /   /___/   14.2   Copyright 1985-2015 StataCorp LP
  Statistics/Data Analysis            StataCorp
                                      4905 Lakeway Drive
     MP - Parallel Edition            College Station, Texas 77845 USA
                                      800-STATA-PC        http://www.stata.com
                                      979-696-4600        stata@stata.com
                                      979-696-4601 (fax)

Single-user 4-core Stata perpetual license:
       Serial number:
         Licensed to:


Notes:
      1.  Unicode is supported; see help unicode_advice.
      2.  More than 2 billion observations are allowed; see help obs_advice.
      3.  Maximum number of variables is set to 5000; see help set_maxvar.

running C:\UTS\profile.do ...

.
```

To access the Variables window, I just click on the tab at the bottom left.

About menus and dialogues The Command window is not the only way of telling Stata what to do. You can also access the Stata commands and their options through a menu and dialogue system. However, we will not use the menu system much in this course.

In econometric and statistical data analysis it is very important that to be able to reproduce results (by yourself or by someone else, possibly at a much later date). This is best guaranteed by writing all the instructions in a 'do' file that you can tell Stata to execute line by line. Anybody with the do file can then rerun the commands and reproduce the results exactly at any time. The Command window is good for playing around with Stata and with data when you are not worried about reproducibility, so for the moment we will continue using the Command window.

Looking ahead: We will talk about do files in a later part of this introduction.

Using Stata as a calculator Stata can work as a calculator using the display *exp* command. Here *exp* stands for a Stata 'expression' which is like a mathematical formula. Try typing display 1+exp(2) in the Command window and hit the Enter key. The output in the Results window should be

```
. display 1+exp(2)
8.3890561
```

Notice that first your command is repeated (with a dot at the start to indicate this line is a command), then the result appears.

This little example shows that Stata knows common mathematical functions such as the exponential. Actually Stata knows a huge number of mathematical and statistical functions. For example, to find the 2.5% percentile of the standard normal distribution, we can use the command display invnormal(0.025). Here invnormal is Stata's name for the normal inverse cumulative distribution function.

Exercise: Find the 2.5% and the 97.5% percentiles of the standard normal distribution using the display command.

What happens when we ask Stata to evaluate an 'illegal' expression? For example, if we ask Stata to compute the logarithm of zero we get

```
. display log(0)
.
```

The answer is a dot!? The dot is Stata's code for 'not a valid number', or a 'missing value'.

Exercise: Try division by 0. Is there a difference between 1/0 and 0/0?

Abbreviating commands Note that Stata commands are case-sensitive, so `display` is not the same as `Display`. Many commands can be abbreviated using the first few characters of their name. In this case `dis` or even `di` works (but `d` expands to `describe`). Abbreviating commonly used commands saves you a lot of typing, but too much abbreviation can make it difficult to understand what is going on when you later look at the output, so it is better not to overuse this feature. (What was it now that '`d dr`' meant?)

Editing in the `Command` window It is possible to cycle through previous instructions in the `Command` window using the `PageUp` and `PageDown` keys. The usual navigation keys like the arrows ←, ↑, →, ↓, and the `Insert` and `Delete` keys all work as expected. For example, this is handy if you make a mistake, because you can retrieve the previous instruction with `PageUp`, navigate to and fix the error using the arrows and the `Insert` and `Delete` keys, and reissue the amended instruction with `Enter`.

Stata documentation and online help The online help is hugely useful, and very simple to access. Simply type `help` *command_name* in the `Command` window and Stata will show the relevant help page in a separate window called the `Viewer`. (In the command examples in this document, italics indicate something that you should not type verbatim; see the exercise below.) Some commands can do a great many different things, and the help page can therefore be quite long. For a beginner, the amount of information can seem overwhelming, but with experience you will learn to identify the bits that you need and ignore the rest.

In addition to the help pages, there is a very extensive information in the Stata PDF documentation files about Stata itself, about all the commands, and in some cases also about econometrics. These files have all the details about the commands and their options, often many examples from econometrics and statistics, as well as the exact technical details about what is actually computed (the formulas). You can access the Stata documentation using the menu system with `Help > PDF documentation`, or you can use the links at the top of the online help pages.

If you are not quite sure what the topic is, try `search` *topic* and Stata will suggest commands for you.

Exercise: Try `help display`, `help invnormal`, and `help search`.

Closing Stata To close Stata, the usual Microsoft Windows methods work, such as selecting `File > Exit` in the menu, or pressing `Alt-F4`. You can also close Stata from the `Command` window with the command `exit`.

If a dataset is loaded, Stata may refuse to close with the message 'no; data in memory would be lost'. This happens if there are unsaved changes to the dataset. If you don't mind losing those changes, use the command `exit, clear`.

Commands and options The instruction `exit, clear` is an example of a command with an option. Many Stata commands can do things in different ways. There is a default way when no option is specified (eg `exit` causes an error message), and another way which is specified after a comma (eg `exit, clear` clears the memory and closes Stata without error). We will see many more examples of this shortly.

3 Data management: the Stata dataset

One dataset at a time Stata always works on one dataset at a time. You can think of the dataset as a matrix where the columns are variables and the rows are observations. Once the dataset is loaded, you can see the variable names in the `Variables` window and some of their properties in the `Properties` window. The `Data Editor`, which you can access via the menu system by selecting `Window > Data Editor`, shows the data in a spreadsheet-like format. (You can also bring up the `Data Editor` by issuing a `browse` command.) Different versions of Stata have different limits on how big a dataset they can handle.

Exercise: Try `help limits` to see how many observations and variables your system allows.

The working directory When you analyse data, you will soon need to read and write files of different kinds: data files, log files, graphs, etc. Unless otherwise indicated, Stata will read and write files to and from the 'working directory'. (This is similar to the 'default folder' in other Windows programs.) To see where Stata thinks this is, type `cd` or `pwd`. To see what files are there, type `dir` or `ls`. (No point in trying to explain these cryptic command names now — they go back to ancient times when computers were as big as houses.)

When you have just started Stata, chances are that the current working directory is not where you want to work. To change the working directory, type `cd` *`directory_name`* or use the menu system with `File > Change working directory`.

```
. cd "D:\EMET8005\Week01"
D:\EMET8005\Week01

. dir
  <dir>   2/13/18  8:47  .
  <dir>   2/13/18  8:47  ..
 551.4k   2/13/18 15:19  NHISmodif.dta
```

If you have spaces in the directory name, you must enclose it in double quotes; otherwise they are optional. Stata accepts both \ or / in the directory name, as well as using the double backslash \\ to point to a server.

Loading a dataset Stata can read data in many formats. Stata's own format is stored in files with a '.dta' extension. Stata data files are loaded with the `use` *`filename`* command. If Stata complains it can't find this file, then you can either reset the working directory or, if the file is not in the working directory, you can specify the whole file path.

Exercise: If you haven't already, download the file NHISmodif.dta from the Wattle course site, fire up Stata, and try `use "NHISmodif.dta"`. Try also specifying the full file path. On my system, I can use the command `use "D:\EMET8005\Week01\NHISmodif.dta"`, but you need to substitute the path relevant on your system.

Here is what happens in my `Results` window:

```
. use "NHISmodif.dta"
(Based on NHIS_clean.dta from Angrist and Pischke)
```

The double quotes are mandatory if the file name contains spaces, otherwise they are optional. The extension '.dta' is also optional. The message 'Based on . . . ' in parentheses is the 'data label' which provides information about the dataset. Not all datasets have a data label.

Looking ahead: A later part of this introduction will explain how we can read data files in other formats (using `import` or `infile`) and how we can create a data label (using `label data` *text*).

Saving a dataset Stata can also save data in many formats. Stata's own format is stored in files with a '.dta' extension. Stata data files are saved with the `save` *filename* command. For example, try `save test.dta`. (If the file already exist, you need `save test.dta, replace` to overwrite it. This is another example of how an option can modify a command in Stata.)

```
. save "test.dta"
file test.dta saved
```

Double quotes are mandatory if the file name contains spaces, otherwise they are optional. The extension '.dta' is also optional. We've seen this a few times. You get the picture.

Looking ahead: A later part of this introduction will explain how we can write data files in other formats (using `export` or `outfile`).

Clearing the dataset from memory You can use `clear` to tell Stata to unload the dataset currently in memory. This is useful if you want to look at a different dataset from the one currently loaded, since Stata only allows one dataset in memory at a time.

```
. clear
```

Describing the dataset To see what data are loaded, you can check the `Variables` window or you can use the `describe`. The command will produce a table of the file name, the number of observations, the number of variables, and a list of all the variables and some of their (non-statistical) properties.

```
. use NHISmodif.dta
(Based on NHIS_clean.dta from Angrist and Pischke)

. describe

Contains data from NHISmodif.dta
  obs:        18,790                          Based on NHIS_clean.dta from
                                                Angrist and Pischke
 vars:            12                          13 Feb 2018 14:47
 size:       789,180
```

```
--------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
--------------------------------------------------------------------------------
serial          long    %12.0g                Sequential Serial Number,
                                                Household Record
pernum          byte    %8.0g                 Person number within family (from
                                                reformatting)
year            float   %8.0g                 Year
gender          str6    %9s                   Gender
racenew         byte    %9.0g      racenew_lbl
                                              Self-reported Race (Post-1997 OMB
                                                standards)
age             byte    %9.0g      age_lbl    Age
inc             float   %9.0g                 Income
famsize         byte    %9.0g      famsize_lbl
                                              Number of persons in family
hlth            float   %9.0g                 Health status (1 poor ... 5
                                                excellent)
hi              float   %9.0g                 Some health insurance 1, none 0
hhweight        long    %12.0g                Household weight, final annual
perweight       double  %9.0f                 Final basic annual weight
yedu            float   %9.0g                 Years of education
--------------------------------------------------------------------------------
Sorted by: serial  pernum
```

In these data, each observation corresponds to a particular person in the survey. You can see that there are 18790 observations and 12 variables. The size of the dataset is about 0.8 MB. The data label appears, as well as the date the file was created. A brief introduction to the variable properties follows.

`Variable name` Stata variable names can be up to 32 characters long, but most people find that short names make life easier. So be prepared to see a lot of short somewhat obscure abbreviations. Note that Stata names are case sensitive, so `income` and `Income` and `IncomE` are different variables.

`Storage type` Stata variables can be either numbers or strings (ie text), just like in a spreadsheet program. Unlike a spreadsheet program Stata can store numerical variables more compactly using different storage formats for integers (bytes, integers, or longs) or floating point numbers (float or double). These data types differ in how much memory they take up and in the range and precision of the values they can represent. String variables can have varying length up to a maximum that depends on which version of Stata you are using. In `NHISmodif.dta` dataset described above, 'str6' indicates that `gender` is a string variable with maximum length 6 characters.

`Display format` The formats used to display the data values (eg in the `Data Editor` and the `Results` windows) are indicated in the column 'display format'. The syntax is kind of cryptic, but essentially they specify how many character places to reserve in total (including leading blanks), and how many digits after the decimal point to show.

`Value label` Often in econometrics, we need to assign numerical codes to categorical variables. Value labels provide a way to assign a description to these numerical codes. For example, 1 may represent ‘Males’ and 2 represent ‘Females’. As we will see later, Stata will use these descriptions for example when it creates tables and figures.

`Variable label` Each Stata variable may have an associated variable label which provides additional information about the variable. This is very a useful feature, and absolutely essential if the variable names themselves are not obvious.

Looking ahead: The variable properties, and how to manipulate them, are discussed in more detail in a later part of this introduction.

Wildcards When referring to a list of variable names, the use of ‘wildcards’ is often useful. Stata understands two. The question mark `?` stands for a single character. For example, in `ag?` the question mark can be any letter or number, so in general `ag?` would include `agA`, `aga`, `agb` etc. Similarly, `a??` expands to all variables in the current dataset whose name begins with `a` and is three characters long. As it turns out, in the `NHISmodif.dta` dataset there is only one, namely `age`.

```
. use NHISmodif.dta
(Based on NHIS_clean.dta from Angrist and Pischke)

. des a??

              storage   display    value
variable name   type    format     label      variable label
--------------------------------------------------------------------------------
age             byte    %9.0g      age_lbl    Age
```

The asterisk `*` stands for any bit of text (including nothing). For example, `*en*` will include all variables that contain `en`.

```
. describe *en*

              storage   display    value
variable name   type    format     label      variable label
--------------------------------------------------------------------------------
gender          str6    %9s                   Gender
racenew         byte    %9.0g      racenew_lbl
                                              Self-reported Race (Post-1997 OMB
                                                standards)
```

Wildcards can be used anywhere you need to refer to a list of variables.

Exercise: What do you think `list *` does? Load the data and try it. What happens if you run `list a??`? Why?

Listing data: `list` To list the observations for all variables, one screen at a time, use the `list` command. If there are too many variables, the lines will be wrapped so it is sometimes better to use `list` *`varlist`*, where *`varlist`* is a list of one or more variable names (separated by spaces).

```
. list serial pernum year gender racenew age inc

     +-----------------------------------------------------------------+
     | serial   pernum   year   gender     racenew   age         inc |
     |-----------------------------------------------------------------|
  1. |      3        1   2009   Female       White    29    19.28293 |
  2. |      3        4   2009     Male       White    35    19.28293 |
  3. |      5        1   2009                    .     .           . |
  4. |      5        2   2009                    .     .           . |
  5. |     10        1   2009     Male       White    45    85.98578 |
     |-----------------------------------------------------------------|
  6. |     10        2   2009   Female       White    44    85.98578 |
  7. |     17        1   2009     Male       White    49    167.8445 |
--Break--
r(1);
```

Note that you can also look at (and edit) the data using the `Data Editor`, which is accessed via the menu system using `Window > Data Editor`.

Stopping command execution Normally Stata will display a screen worth of data at a time and wait for you to press the `SpaceBar` before showing the next screen (indicated by `--more--` in the `Results` window). If you have seen enough and don't want the rest of the output, click on the `Break` button ⊗ in the menu. Or you can press `Ctrl-K` to 'kill' the current listing command (causing `--Break--` and the 'error' message `r(1);` to appear in the `Results` window). You can also turn the waiting off completely with the command `set more off`, which must be issued beforehand and affects all subsequent commands. You guessed right — to reset waiting you simply issue the command `set more on`.

Missing values Note the 'missing values' (empty strings or dots) in the `gender` and `age` variables in the list above. Stata has many special codes for missing values, namely `.`, `.a`, . . . , `.z` for numerical variables and the empty string for string variables. (Note the single '`.`' is a Stata value, while '. . . ' is the usual ellipsis. Confusing? Perhaps.) The different codes can be used to distinguish between different kinds of missing values. This is useful for example for data managers who might use `.a` when the question wasn't asked, `.b` when the survey respondent didn't know the answer, `.c` when the respondent refused to answer, etc. In practical analysis, the reason why a value is missing rarely matters, so in most cases missings are simply coded with the simple dot.

Looking ahead: With one exception that we will get to shortly, Stata is pretty good at handling missing values in a sensible way.

The `in` qualifier Many Stata commands that process observations (such as the `list` command but not the `display` command) allow you to select a subset of observations for processing. For example, it is possible to list a specific range of observations using the `in` *`range`* qualifier. The *`range`* is given as *`a/b`*, where *`a`* is the first row and *`b`* the last row to be processed. For example, we can list only observations from 25 to 27 as follows.

```
. list serial pernum year gender racenew age inc in 25/27

     +-------------------------------------------------------------+
     | serial   pernum   year   gender   racenew   age         inc |
     |-------------------------------------------------------------|
 25. |     46        1   2009   Female     White    50    167.8445 |
 26. |     46        2   2009     Male     White    52           . |
 27. |     47        1   2009     Male     White    31    85.98578 |
     +-------------------------------------------------------------+
```

The letters *f* and *l* can be used in the range to denote the first and the last observation respectively, and negative numbers may be used to specify distance from the end of the data.

Exercise: Load the dataset and list the last 5 observations. Confused? Then try `help in` to see some examples.

The `if` qualifier Another way of selecting a subset of observations for processing uses a logical test, and only processes the observations for which the test is true. For example, it is possible to list specific observations using the `if` *exp* qualifier. Here *exp* is an expression that evaluates to either true or false for each observation in the dataset. For example, we can list only observations for females as follows.

```
. list serial pernum year gender racenew age inc if gender=="Female"

       +-------------------------------------------------------------+
       | serial   pernum   year   gender     racenew   age       inc |
       |-------------------------------------------------------------|
    1. |      3        1   2009   Female       White    29   19.28293 |
    6. |     10        2   2009   Female       White    44   85.98578 |
    8. |     17        2   2009   Female       White    55   167.8445 |
   10. |     19        2   2009   Female       White    43   61.10297 |
   12. |     24        2   2009   Female       White    34   85.98578 |
       |-------------------------------------------------------------|
   14. |     28        2   2009   Female       White    54   70.83464 |
--Break--
r(1);
```

We will dig deeper into Stata expressions shortly, but note how Stata uses the syntax `==` when we want to know whether the left- and right-hand sides are identical. Also, notice that we need double quotes around ‘Female’. Without the double quotes, Stata would think we are referring to a variable called `Female` and will issue an error message when it cannot find it in the dataset. Usually bits of text must have quotes around them to distinguish them from variables and commands.

Exercise: Load the dataset and list the observations with `age` equal to 44.

Creating/generating new variables You can create new variables by transforming existing ones using the `generate` *newvar*=*exp* command. As for the `display` command, *exp* stands for a Stata ‘expression’ which is like a mathematical formula. The transformations may involve numbers such as multiplying a fractional variable by 100 to get per cent, functions such as taking the logarithm of a variable, and arithmetic operations such as adding variable and fixed costs variables to get total costs. The expression is evaluated separately for each observation in the dataset.

For example, if we are interested in the square of income or the logarithm of income, then we might issue the following commands.

```
. use NHISmodif.dta
(Based on NHIS_clean.dta from Angrist and Pischke)

. generate inc2=inc^2
(1,863 missing values generated)

. gen lninc=ln(inc)
(1,863 missing values generated)

. describe inc inc2 lninc

              storage   display    value
variable name   type    format     label      variable label
--------------------------------------------------------------------------------
inc             float   %9.0g                 Income
inc2            float   %9.0g
lninc           float   %9.0g

. list inc inc2 lninc

       +--------------------------------+
       |      inc       inc2      lninc |
       |--------------------------------|
    1. | 19.28293   371.8315    2.95922 |
    2. | 19.28293   371.8315    2.95922 |
    3. |        .          .          . |
    4. |        .          .          . |
    5. | 85.98578   7393.554   4.454182 |
       |--------------------------------|
    6. | 85.98578   7393.554   4.454182 |
    7. | 167.8445   28171.79   5.123038 |
    8. | 167.8445
--Break--
r(1);
```

Note that new variables `inc2` and `lninc` now appear. Note also that when an expression involves a variable with missing values, the result is a missing value.

Exercise: Verify that the square of 19.28293 is 371.8315 using the `display` command.

The `generate` command processes observations and it accepts the `if` *`exp`* and `in` *`range`* qualifiers similar to the `list` command.

Exercise: Load the dataset and create a variable containing the birth year of each person, but (for the sake of this exercise) limit the computation to persons with 12 years of education (ie `yedu==12`). What values are assigned for persons with more or less education?

Note that Stata uses `=` for assigning values to the left-hand side in the `generate` command, while `==` is used for relational equality, which evaluates to either true or false.

Choosing sensible variable names can help make a program more readable and life easier. Different people use different systems. For example, for the log of income some might like `lninc` or `loginc`, others insert an underscore `ln_inc`, or use camel-casing `lnInc`.

Modifying/replacing old variables The command `generate` *newvar=exp* is reserved for creating new variables, and Stata will refuse to create a variable if another with the same name already exists.

To modify an existing variable, you can use the command `replace` *oldvar=exp* `in range if exp`. It works exactly the same as `generate`, only Stata will complain if *oldvar* does *not* already exist.

To change the name of a variable without affecting its values, use the command `rename` *oldvar newvar*.

Exercise: Load the dataset if you haven't already done so, and summarise income with `summarize inc`. It looks like income may be measured in 1000 dollars. Replace `inc` with itself multiplied by 1000 to give the values in simple dollars. Summarise the data again to check.

Exercise: Load the dataset and rename `age` to `AGE`.

Dropping variables So it is possible to create new variables and replace existing ones. If you want to delete some variables altogether, use either the `drop` *varlist* or the `keep` *varlist* commands. They are pretty self-explanatory. Try it to see (with `describe`) what happens.

Exercise: Load the dataset and drop all variables that begin with 'r'. Check before/after using `describe`.

4 Stata expressions: functions, operators

Evaluation of expressions Expressions that involve only numbers evaluate to a single number. For example, we have seen that the `display` *exp* command evaluates expressions of this kind

```
. display 1+exp(2)
8.3890561
```

In general, expressions can be formulae that involve variables with many observations (rows in the dataset). Such expressions are evaluated row-by-row. That is, the result of the expression is a number for each observation in the data set. For example, the `generate` *newvar=exp* command evaluates expressions of this kind. We have already seen how to compute the square and the log of the variable `inc`. Here is another example:

```
. generate birthyear=2009-age

. list year age birthyear

       +-----------------------+
       | year   age   birthy~r |
       |-----------------------|
    1. | 2009    29       1980 |
    2. | 2009    35       1974 |
    3. | 2009     .          . |
    4. | 2009     .          . |
    5. | 2009    45       1964 |
       |-----------------------|
    6. | 2009    44       1965 |
--Break--
r(1);
```

Looking ahead: It is also possible to write expressions that combine information across observations, but this is an advanced skill and sometimes very tricky.

Stata functions Stata has many different kinds of functions. You can see them all with `help functions`, but at this point it is better just to list a few of the most important mathematical and statistical ones. Try `help mathematical functions` and `help statistical functions` to see the complete selection.

abs(*x*)	the absolute value of *x*
exp(*x*)	the exponential function of *x*
int(*x*)	the integer obtained by truncating *x* towards zero
round(*x*)	the whole number nearest to *x*
sqrt(*x*)	the square root of *x* if *x*>=0
ln(*x*)	the natural logarithm of *x* if *x*>0
log(*x*)	the natural logarithm of *x* if *x*>0
normalden(*x*)	the standard normal density at *x*
normalden(*x*,*m*,*s*)	the normal density with mean *m* and standard deviation *s* at *x*
normal(*x*)	the standard normal cumulative distribution at *x*
invnormal(*x*)	the inverse standard normal cumulative distribution at *x* if 0<*x*<1

Exercise: Load the dataset and create a new variable that is income rounded to nearest whole dollar.

Exercise: What is the density of a normally distributed random variable with mean 1 and variance 2 evaluated at 1? What is the probability that this random variable takes a value less than 0? (*Hint:* Use the `display` command.)

Looking ahead: There are also functions to work with dates and times, see `help date functions`, and with strings, see `help string functions`.

Relational and logical operators Some functions evaluate the truth of their arguments. For example, the expression `age>=30` will be true for some observations and false for others. The expression `age<30&hi==1` will be true for observations that satisfy both `age<30` and `hi==1`, and false otherwise. That is, it will be true only for persons in the data who are less than 30 years old *and* also have health insurance. Similarly, the expression `age<30|hi==1` will be true for observations that satisfy either `age<30` or `hi==1` or both, and false otherwise.

One of the most important uses logical tests is with the `if` *exp* qualifier, as we saw earlier. This allows you to select a subset of observations for processing. This can also be used to create new variables. As an example, while our dataset has a variable for income, namely inc, maybe we just want to know how many households are rich and how many are poor. For concreteness, let's suppose the cut-off is $20000. To create a variable that is 1 if a person is poor and 0 otherwise, you can begin by letting poor be 0 for everyone and then change the values to 1 for those who are poor.

```
. generate poor=0

. replace poor=1 if inc<20

. list serial pernum inc poor

       +----------------------------------+
       | serial   pernum        inc   poor |
       |----------------------------------|
    1. |      3        1   19.28293      1 |
    2. |      3        4   19.28293      1 |
    3. |      5        1   167.8445      0 |
    4. |      5        2   167.8445      0 |
--Break--
r(1);
```

Technically, computers generally represent truth as 1 and the opposite ('untruth'? 'falseness'? 'lies'?) as 0. With a little creativity, this can be utilised in very powerful ways. The instruction generate poor2=inc<20 may look very weird and not mathematically meaningful at first, but the computer doesn't care. It will evaluate the right-hand side to either 0 (false) or to 1 (true), observation by observation, and these values will be assigned to the new variable poor2. The result is exactly the same as the earlier definition of poor. Some programmers find the first way more intuitive. Others prefer the latter because it is more compact. This kind of coding is ubiquitous in programming, not just with Stata.

Exercise: Load the data and create both poor and poor2. Use list to check that the two variables are the same.

Verifying properties Stata has a useful little command, assert *exp*, that allows you to verify whether something is correct (for all observations) or not. If the expression evaluates to false (ie 0), Stata issues an errors message and halts execution. If the expression evaluates to true (ie 1), Stata just continues. Note the expression must be true for all observations to be true overall.

Exercise: Use assert to verify that poor and poor2 are identical. (*Hint:* Recall the relational equivalence operator is ==.)

Overview of operators Typing help operators brings up a table like this:

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

The help page also informs us that the order of evaluation (from first to last) of all operators is ! (or ~), ^, - (negation), /, *, - (subtraction), +, != (or ~=), >, <, <=, >=, ==, &, and |. If you want something else, use parentheses.

Exercise: Think about the truth value of the expressions `age<30|age>20&gender=="Female"`, `age<30&age>20|gender=="Female"`, and `age<30&(age>20|gender=="Female")`. Then use `list` to check your thinking.

As already mentioned, Stata uses a single equal sign (=) for assigning a value to a left-hand side variable name and a double equal sign (==) for the relational equality test.

5 Data analysis: univariate statistics

Sample statistics: means, variances One of the most useful things to do to get a feel for the data is to have a look at the means, variances, and range of the data. The `summarize` command (abbreviated `sum` or even `su`) is a quick way to accomplish this. By default `summarize` considers all variables in the dataset. If you only want to look at a subset you can use `summarize` *`varlist`*.

```
. use NHISmodif.dta, clear
(Based on NHIS_clean.dta from Angrist and Pischke)

. sum

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
      serial |     18,790    20422.78    11913.15          3      41173
      pernum |     18,790    1.523257    .5617605          1          9
        year |     18,790        2009           0       2009       2009
      gender |          0
     racenew |     18,788    13.98233    9.286123         10         50
-------------+---------------------------------------------------------
         age |     18,788    42.72003    8.698511         26         59
         inc |     16,927    94.23244    56.71635   19.28293   167.8445
     famsize |     18,790    3.633209    1.369825          2         18
        hlth |     18,790    3.932464    .9526357          1          5
          hi |     18,790    .8417243     .365009          0          1
-------------+---------------------------------------------------------
    hhweight |     18,790    2981.065    1951.404        724      26014
   perweight |     18,790    3546.153      2323.4        719      31494
```

There are a few things to note here. First, most of the variables have 18790 observations. Here 'Obs' actually indicates the number of non-missing numerical observations, so the results say that the numeric variables with missing value are racenew, age, and inc. The reason there are 0 observations for gender is simply that it is a string variable and summarize only considers numeric variables.

Second, the mean and the standard deviation ('Std. Dev.') only make sense for variables whose values can be meaningfully added. Addition is not meaningful for id numbers such as serial, since household number 5 plus household number 11 does not give household number 16. The same is true for the person number pernum (which is uniquely defined within the household only). Stata doesn't know and doesn't care that these means are meaningless, so computes and reports them as always. It is up to you to interpret the numbers correctly. Note though that even for serial it may be useful to know that the minimum is 3 and the maximum is 41173. Obviously, there must be some gaps in the sequence, since the total number of observations is only 18790.

Exercise: Interpret the summary statistics for year and age.

Exercise: The summarize command has an option detail which is often useful. Compare sum famsize and sum famsize, detail

The in and if qualifiers The qualifiers also work here, as they do for most Stata commands that process observations. For example, we can get summary statistics computed for observations 25 through 30 only:

```
. sum serial pernum year gender racenew age inc in 25/27

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
      serial |          3    46.33333    .5773503         46         47
      pernum |          3    1.333333    .5773503          1          2
        year |          3        2009           0       2009       2009
      gender |          0
     racenew |          3          10           0         10         10
-------------+---------------------------------------------------------
         age |          3    44.33333    11.59023         31         52
         inc |          2    126.9152    57.88288   85.98578   167.8445
```

Or, possibly more usefully, we can get summary statistics for males and females separately. For example:

```
. sum serial pernum year gender racenew age inc if gender=="Female"

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
      serial |      9,394    20424.95    11912.24          3      41173
      pernum |      9,394    1.612412    .5442021          1          7
        year |      9,394        2009           0       2009       2009
      gender |          0
     racenew |      9,394    14.06536    9.404973         10         50
-------------+---------------------------------------------------------
         age |      9,394    41.74675    8.647235         26         59
         inc |      8,516    94.10039     56.6731   19.28293   167.8445
```

Exercise: Compute summary statistics for the years of education (yedu) for males and females separately and compare.

Missing values Most of Stata's statistical commands handle missing values by 'casewise deletion'. For example, the mean of income is calculated simply by omitting the missing values (in both the numerator and denominator).

```
. summarize inc

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
         inc |     16,927    94.23244    56.71635   19.28293   167.8445

. summarize inc if !missing(inc)

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
         inc |     16,927    94.23244    56.71635   19.28293   167.8445
```

Notice that output for the two commands is the same, so Stata must have omitted the missing in first case without telling you about it.

More generally, if Stata is about to do some computation involving variables x and y, it will simply ignore all observations with missing values in either of those two variables. (Missing values in other variables are not checked.)

Frequency tables Many datasets include discrete and categorical variables. For these, frequency tables are often very informative. To tabulate the race distribution in the NHISmodif.dta dataset, use tabulate racenew.

```
. tabulate racenew, missing

 Self-reported Race (Post-1997 |
                OMB standards) |      Freq.     Percent        Cum.
-------------------------------+-----------------------------------
                         White |     15,060       80.15       80.15
        Black/African American |      1,875        9.98       90.13
American Indian/Alaskan Native |        146        0.78       90.90
                         Asian |      1,513        8.05       98.96
                Multiple Race |        194        1.03       99.99
                             . |          2        0.01      100.00
-------------------------------+-----------------------------------
                         Total |     18,790      100.00
```

So we see that about 80% of this sample is 'White', 10% is 'Black/African American', 8% is 'Asian' (maybe Asian American?), and the remaining 2% are either native or mixed or unknown. (Note that Hispanics in the US can be any race, although the majority identify as White. From 2000, the Census has a separate category for Hawaiian/Other Pacific Islanders, and in 2014 the Middle Eastern/Arab Americans got split off from the Whites.)

Without the option missing, tabulate will omit the missing values in the calculations. So the total will only be 18,788 observations.

Tables and value labels Using describe racenew, we can see that racenew is actually a numerical variable (storage type byte). The row headings in the table above are coming from the value labels in racenew_lbl. If you want to see the underlying numerical codes, use the nolabel option.

```
. describe racenew

              storage   display    value
variable name   type    format     label      variable label
--------------------------------------------------------------------------------
racenew         byte    %9.0g      racenew_lbl
                                              Self-reported Race (Post-1997 OMB
                                                standards)

. tabulate racenew, nolabel

Self-report |
    ed Race |
 (Post-1997 |
        OMB |
 standards) |      Freq.     Percent        Cum.
------------+-----------------------------------
         10 |     15,060       80.16       80.16
         20 |      1,875        9.98       90.14
         30 |        146        0.78       90.91
         40 |      1,513        8.05       98.97
         50 |        194        1.03      100.00
------------+-----------------------------------
      Total |     18,788      100.00
```

Maybe you now see why value labels are so useful?

Exercise: Tabulate family size (famsize), and comment on the distribution. Compute also the mean and the standard deviation for family size and for race. Contemplate their meaning.

Estimating a population mean Describing what is in the data is important, but ultimately we want to do more. We want to use the data in the sample to make inference about the population. The mean *varlist* command will compute an estimate of the population mean and provide a standard error and a confidence interval as well.

```
. tab hi

Some health |
  insurance |
  1, none 0 |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |      2,974       15.83       15.83
          1 |     15,816       84.17      100.00
------------+-----------------------------------
      Total |     18,790      100.00
```

```
. mean hi

Mean estimation                         Number of obs = 18,790

--------------------------------------------------------------
             |       Mean   Std. err.     [95% conf. interval]
-------------+------------------------------------------------
          hi |   .8417243   .0026628       .836505    .8469437
--------------------------------------------------------------
```

So hi is 1 for people with some private health insurance cover and 0 for people without. The tabulate and mean commands both agree that 84.17% of people have insurance. The mean commands also provides a standard error, which is very small, and a 95% confidence interval, which is very narrow. The precision of this estimate is great, as can be expected when the sample size is so large.

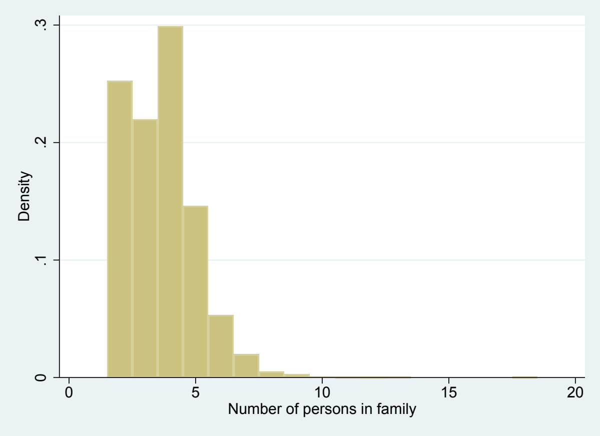
A caveat is warranted here. Although this estimate is in the ballpark, it is not a good estimate of the population mean. The problem is that the NHIS dataset is not a simple random sample, so we need some adjustment. We will discuss this issue in more detail in a later part of this introduction.

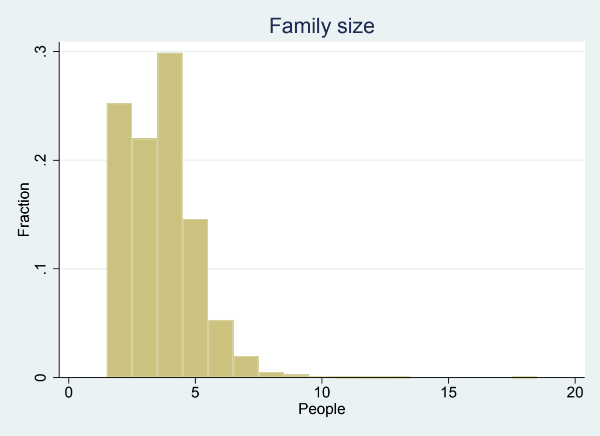
6 Stata graphics: histograms

Histograms Another way to examine the distribution of family size is to draw a histogram. Conveniently, Stata has the command histogram *varname*. This command does not produce much output in the Results window; instead a nice colourful graph pops up in the Graph window. By default, histogram assumes that varname is a continuous variable and it will summarise the distribution using a small number of 'bins'. For discrete variables, we can plot the exact distribution using the discrete option.

```
. histogram famsize, discrete
(start=2, width=1)
```

The result is shown in the left figure below.





Note that the label on the horizontal axis is taken from famsize's variable label. You can overwrite this using the option xtitle. You can also overwrite the vertical axis label with ytitle, or add an overall title to the graph using the option title. For example:

```
. hist famsize, discret xtitle("People") ytitle("Fraction") title("Family size")
(start=2, width=1)
```

The result is shown in the right figure above. Stata graphs are highly customisable and there are a very great many more options to discover later on.

Exporting the graph Having created the graph, you can save it for later use in other applications using the command `graph export` *`filename`*. If a file with the same name already exist, Stata will not overwrite it unless you specify the `replace` option. The file format of the graph (such as .png or .pdf) is determined by the extension of *`filename`*.

```
. graph export NHIS_histogram_famsize.png, replace
(file NHIS_histogram_famsize.png written in PNG format)

. graph export NHIS_histogram_famsize.pdf, replace
(file NHIS_histogram_famsize.pdf written in PDF format)
```

Exercise: Create a graph, export it, and see if you can get it into Microsoft Word and looking nice. (Or into whatever text processor you use for report writing.)

7 Wrap-up

The Stata language Many Stata commands follow the basic syntax

```
command [varlist] [if exp] [in range] [, option_list]
```

where *`command`* denotes a Stata command, *`varlist`* denotes a list of variable names, *`exp`* denotes an algebraic expression, *`range`* denotes an observation range, and *`options`* denotes a list of options. The square brackets distinguish optional qualifiers and options from required ones. Obviously, not all commands are compatible with all syntax components. For example, the `if` and `in` qualifiers do not make sense for the `describe` command.

You can think of the *`command`* as the equivalent of a verb, and the *`varlist`* as the object in this 'sentence'. The qualifiers `if` and `in` specify which observations are used. The *`options`* modify the command. As mentioned, Stata has a default way of doing things, and the options are only needed if you want to change that. Note that there is only one comma in each Stata instruction separating the main instruction from the options; if several options are needed, they are separated by blank spaces. Note also that sometimes the *`command`* is actually two words (as in `graph export`).

There is more to the Stata language than this, some of which you will see in later parts of this *Introduction to Stata* (eg command prefixes, sample weights).