Using Stata Effectively: Data Management, Analysis, and Graphics Fundamentals

Bill Rising
StataCorp LP

Washington, DC
April 9–10, 2009

Contents

Lesson 1

Getting Familiar with Stata

1.1 Introduction

1.1.1 Goals

Goals

- Learning Stata's Interface
- Learning different ways to work in Stata
- Making it a habit to keep logs
- Learning what Stata keeps in a dataset
- Learning some ways to save time

1.2 Basic Stata Usage

1.2.1 The Fundamentals

Stata's Tenets

- Type a little, get a little
  - Click a little, get a little works fine, also!
- Simple reproducibility
- Easy and complete extensibility
- Easy code sharing
- Web awareness

Various Ways to Use Stata

- Menus
  - Menus create dialog boxes, which then create and issue a command
- Command Window
  - The Command window seems more cumbersome at first, but is quite quick for commonly-used commands
- Via Help
  - Help files often link to the desired dialog box. This is a good learning method
- Using a mixture of approaches is fine, especially for new commands
  - When automating Stata, this becomes less useful, but we'll see ways to mix techniques even then

---

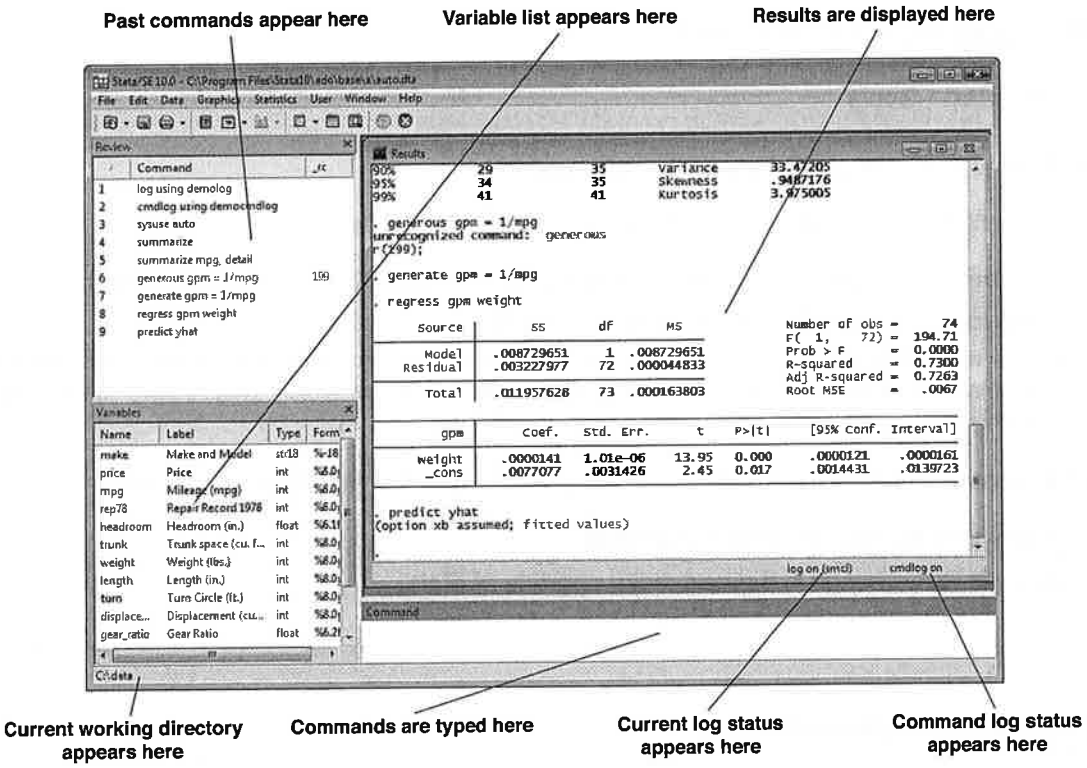
Some Typography

- `Items which must be typed as shown` will be in a `monospaced font`
- *Items for which a substitution is needed* will be in *italics*
- [`Optional items`] will be [`in square brackets`] (though the brackets do not get typed)
- **Menu items** will be in **bold**.
  - Submenus will be indicated with **>**, as in **Main Menu > Submenu Item**
- Example Stata commands will often be preceded by a `.`
  - The `.` is a prompt and does not get typed—it is for distinguishing input from output in the notes

---

Starting Stata

- Start Stata from the Desktop or the Start menu



Keeping Up-to-date

- By default, Stata checks periodically to see if there are any updates available
  - If this doesn't happen here, type

    ```
    . update query
    ```
- You should keep Stata up-to-date
  - The updates within each numbered version are free and easy enough to install

Let's Customize!

- Before starting, you might want to change the look of Stata's windows
- Go to the typical place for a **Preferences** menu
  - Under the **Edit** menu on MS Windows and Unix
  - Under the *Stata* menu on the Mac (where *Stata* is the version and flavor of Stata you are running)
- Under the **General Preferences** you can change the colors and font sizes in the windows
- Under the ***Internet*** tab, you can turn on automatic update checking
  - This is worthwhile, unless you often off the internet

1.2.2 Keeping Track of Your Work

Keeping Organized—Goals

- Would like to have one place for everything in a project
- Since everything is in one place, we would like Stata to know about it
  - We'll be typing many commands, even some for files, and this will save many painful keystrokes
  - In the long run, this will make our project portable
    When we get to automation, we'll see that by working within our project folder, we can move the folder from place to place, and everything will move with it and just work. This is very handy, especially when passing the project off to another person—your future self.
- Another reason for this is Stata's default working directory in MS Windows: `c:\data`
  - A very poor location for shared machines
  - This will likely change in the next major upgrade of Stata...

Keeping Organized—Implementation

- We need to create a new folder to hold our project—the notes from this lesson
  - Click **File > Change Working Directory** ...
    This menu item exists in Stata 9 for the Mac and all versions of Stata 10. It does not, unfortunately, exist in Stata 9 for Windows.
    If you are using an older version of Stata without the menu item, you need to do some typing (possibly quite a lot).
    You will need to type:
    `cd "`*c:/the/full/path/to/your/folder*`"`
    What this actual path is will depend heavily on your Windows installation and whether you are working in a lab. By default, Stata for Windows starts working in the directory `c:/data`. This is not the best place to be working in shared environments, so we'll cover what path to use in class.
    Note: The forward slashs are intentional. Stata understands both slashes and backslashes—and the latter can sometimes cause trouble when programming. Stata can even understand a mixture of the two, so there is nothing to worry about.
  - Click the Desktop icon
  - Click the **Make New Folder** button
  - Type `startup` without changing the focus
  - Click the **OK** button
- Note the resulting command:`cd startup`
- In general, you would not have to make a new folder each time—we will be doing it here to mimic separate projects

---

Opening/Creating a Log File

- From the Stata GUI
  - Click the Log... button
  - Navigate to the `startup` folder
  - Save a log file named `startup` in your new folder
  - The resulting command shows the full path to the log file
- Using the Command window
  - Typing `log using "startup"` in the Command window would have worked, also
    * The quotes are optional when there are no spaces in the filename.
    * Stata does not need the whole path, because we changed the working directory to the proper location in the preceding step.

Log File Notes

- Log files contain everything which appears in the Results window (after the log file is opened, of course)
- By default, they are written to match the Results window, including clickable links and the like
  - The log files are written as smcl (Stata Markup and Control Language) files—readable by Stata and messy elsewhere, in a similar fashion to web pages
  - They can be translated to plain text if need by using the `translate` command—see `help translate`
- The log file is what it is: a record of everything apppearing in the Results window—nothing more, nothing less

---

Ready to Work

- We are now ready to work
- The log file keeps track of our commands and results
- If we save files, datasets, or graphs, our work will all be in the same place
- Since we are working within the folder, it will be easy to move everything to a new location, if desired

---

1.3 Basics: Interface and Datasets

1.3.1 Stata and Data

Example Datasets & the Viewer

- For the first example, we will open a dataset which is shipped with Stata.
- Go to **File > Example Datasets**...
  - A new type of window: a Viewer window opens—we'll see more about this, because the Viewer is used for on-line documentation and help.
- Click the link to the `Example datasets installed with Stata`
- Click the `use` to the right of `auto.dta`
- Close the Viewer window

What Happened?

- When the file is opened, we are told a little about the dataset.
  - This is an example of a *label* which is attached to the data.
- Stata has read in the dataset into memory, and is now ready to work
  - Stata always reads the entire dataset into memory—more about this later
- The command `sysuse auto.dta` appeared in the Results window.
- The same command appeared in the Review window.
- Variables appeared in the Variables window

Aside: Learning from the Results Window

- Stata echoed the command `sysuse auto.dta` to the Results window
- This means that we could type `sysuse auto.dta` instead of navigating through the menus
  - Because Stata datasets have a `.dta` extension by default, we could have also typed `sysuse auto`
- **Lesson**: one of the best ways to learn commands is to use the GUI and see what pops up in the Results window.

Data as Table

- Press the **Browse** button.
- We can see that Stata views the data as a single *table*.

Looking at the Data

- Change the active cell using the arrow keys or the mouse
  - The `make` variable is displayed in red, because it contains text
    - Stata calls this a *string* variable
  - The commas in the `price` variable do not show in the entry bar
    - This is an example of formatting—the comma is displayed, but it is not stored
  - Some values of `rep78` are displayed as a dot.
    - These are missing values within a numerical variable.
  - The `foreign` variable displays as blue text, but the values are numbers
    - This is an *encoded* variable—it is stored as a number but has *value labels* to make its values human-readable

---

Data *vs.* Dataset

- The above notes point out the difference between *data* and a *dataset*
- The *data* are the raw numbers and words stored in the file.
- A *dataset* is much more than this
  - It contains all sorts of extra information which make the data understandable, along with display formats for the data.
  - A good dataset documents itself well.
- We'll see how to turn data into a dataset in a later lesson

---

1.3.2 Looking Carefully at the Contents of a Dataset

The Best First Step: `codebook`

- The `codebook` command gives details about both structure & contents.
- **Data > Describe data > Describe data contents (codebook)** will bring up a dialog box
  - This is typical behavior for menus...
- We're interested in all the variables, so click the **OK** button.

```
. codebook                                    or    Codebook, compact

-------------------------------------------------------------------------------
make                                                          Make and Model
-------------------------------------------------------------------------------

                  type:  string (str18), but longest is str17

          unique values:  74                       missing "":  0/74

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

               warning:  variable has embedded blanks

-------------------------------------------------------------------------------
price                                                                    Price
-------------------------------------------------------------------------------

                  type:  numeric (int)

                 range:  [3291,15906]                 units:  1
          unique values:  74                     missing .:  0/74

                  mean:   6165.26
             std. dev:    2949.5

            percentiles:        10%       25%       50%       75%       90%
                               3895      4195    5006.5      6342     11385

-------------------------------------------------------------------------------
mpg                                                                Mileage (mpg)
-------------------------------------------------------------------------------

                  type:  numeric (int)

                 range:  [12,41]                      units:  1
          unique values:  21                     missing .:  0/74

                  mean:   21.2973
             std. dev:    5.7855
```

```
      percentiles:        10%       25%       50%       75%       90%
                           14        18        20        25        29

-------------------------------------------------------------------------------
rep78                                                    Repair Record 1978
-------------------------------------------------------------------------------

                  type:  numeric (int)

                 range:  [1,5]                        units:  1
         unique values:  5                        missing .:  5/74

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

-------------------------------------------------------------------------------
headroom                                                  Headroom (in.)
-------------------------------------------------------------------------------

                  type:  numeric (float)

                 range:  [1.5,5]                      units:  .1
         unique values:  8                        missing .:  0/74

            tabulation:  Freq.  Value
                            4  1.5
                           13  2
                           14  2.5
                           13  3
                           15  3.5
                           10  4
                            4  4.5
                            1  5

-------------------------------------------------------------------------------
trunk                                                  Trunk space (cu. ft.)
-------------------------------------------------------------------------------

                  type:  numeric (int)

                 range:  [5,23]                       units:  1
         unique values:  18                       missing .:  0/74

                  mean:   13.7568
             std. dev:    4.2774

      percentiles:        10%       25%       50%       75%       90%
                            8        10        14        17        20

-------------------------------------------------------------------------------
```

```
weight                                                    Weight (lbs.)
-------------------------------------------------------------------------------
                  type:  numeric (int)

                 range:  [1760,4840]                  units:  10
         unique values:  64                       missing .:  0/74

                  mean:   3019.46
             std. dev:   777.194

            percentiles:        10%       25%       50%       75%       90%
                               2020      2240      3190      3600      4060

-------------------------------------------------------------------------------
length                                                    Length (in.)
-------------------------------------------------------------------------------

                  type:  numeric (int)

                 range:  [142,233]                    units:  1
         unique values:  47                       missing .:  0/74

                  mean:   187.932
             std. dev:   22.2663

            percentiles:        10%       25%       50%       75%       90%
                                157       170     192.5       204       218

-------------------------------------------------------------------------------
turn                                                    Turn Circle (ft.)
-------------------------------------------------------------------------------

                  type:  numeric (int)

                 range:  [31,51]                      units:  1
         unique values:  18                       missing .:  0/74

                  mean:   39.6486
             std. dev:   4.39935

            percentiles:        10%       25%       50%       75%       90%
                                 34        36        40        43        45

-------------------------------------------------------------------------------
displacement                                        Displacement (cu. in.)
-------------------------------------------------------------------------------

                  type:  numeric (int)

                 range:  [79,425]                     units:  1
         unique values:  31                       missing .:  0/74

                  mean:   197.297
             std. dev:   91.8372
```

```
      percentiles:        10%       25%       50%       75%       90%
                           97       119       196       250       350

-------------------------------------------------------------------------------
gear_ratio                                                        Gear Ratio
-------------------------------------------------------------------------------

                  type:  numeric (float)

                 range:  [2.19,3.89]                  units:  .01
          unique values:  36                       missing .:  0/74

                  mean:   3.01486
             std. dev:    .456287

      percentiles:        10%       25%       50%       75%       90%
                         2.43      2.73     2.955      3.37      3.72

-------------------------------------------------------------------------------
foreign                                                           Car type
-------------------------------------------------------------------------------

                  type:  numeric (byte)
                 label:  origin

                 range:  [0,1]                        units:  1
          unique values:  2                        missing .:  0/74

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

- This ought to be the first command used whenever you get a new dataset!

Aside: Long Results

- When a result is too long to fit onto one screen, Stata will scroll until the screen is full and then display `--more--`
  - To show just the next line of output, press the **Enter** key.
  - To show the next screen-full of information,
    - Click on the blue `--more--` in the Results window.
    - Click on the green **Go** button
    - Press any other key (other than q)
  - To let all the information scroll up the window without stopping, press the spacebar repeatedly or start typing.
  - To stop the output, click on the red **Break** button or press **q**

Interesting Results

- The `make` identifies observations—74 distinct values, none missing, out of 74 observations
  - This means is a *primary key* for the dataset—a concept we'll see when putting datasets together
- `rep78` has missing values
  - In this case, all are periods (`.`). If there were other types of missing values, this would be noted here.
- Some numerical variables display frequency tables, some have summary statistics
  - `codebook` defaults to displaying summary statistics if there are at least nine distinct values

---

Units

- The `units` result can sometimes be helpful.
  - If the units are strange, there could be odd data in the dataset
- Here:
  - Prices are in single dollars.
  - Stata thinks that the headroom is measured in tenths of inches.
  - Vehicle weights are in tens of pounds.
  - Gear ratios are in hundredths.

Aside: Stata & Missing Values

- Stata understands missing numerical values.
  - When displayed on the screen, they appear as a period or a period with a letter: `.`, `.a`, ... `.z`
  - Missing numerical values are always treated as the largest possible numerical values.
    - Important when sorting or filtering the data!
- Missing string (text) values are just the empty string: ""
  - Can think of a missing string value as a blank.
  - Missing string values always come before all other string values.

Codebook and Large Datasets

- For a dataset with hundreds of variables, the codebook command is still useful, but it is time consuming to pore over the output
- A shorter, but less informative version is available:
  - Open the codebook dialog again
  - Click the ***Options*** tab
  - Click the **Display compact report on the variables** checkbox
  - Click the **OK** button
- We see that the command is simple:

```
. codebook, compact
```

```
Variable      Obs Unique      Mean    Min    Max  Label
-------------------------------------------------------------------------------
make           74     74         .      .      .  Make and Model
price          74     74  6165.257   3291  15906  Price
mpg            74     21   21.2973     12     41  Mileage (mpg)
rep78          69      5  3.405797      1      5  Repair Record 1978
headroom       74      8  2.993243    1.5      5  Headroom (in.)
trunk          74     18  13.75676      5     23  Trunk space (cu. ft.)
weight         74     64  3019.459   1760   4840  Weight (lbs.)
length         74     47  187.9324    142    233  Length (in.)
turn           74     18  39.64865     31     51  Turn Circle (ft.)
displacement   74     31  197.2973     79    425  Displacement (cu. in.)
gear_ratio     74     36  3.014865   2.19   3.89  Gear Ratio
foreign        74      2  .2972973      0      1  Car type
-------------------------------------------------------------------------------
```

Codebook h (and then hit the tab key)

A Good Second Step: The `notes` Command

- Some datasets have notes attached to them
- We can see these by typing

```
. notes
```

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

  - We will learn to attach notes later
- There are not many here

---

db codebook (to get the dialogue box)

1.3.3 Interface: Dialog Boxes and Commands

Looking at Dialog Boxes

- Open the previous dialog, again, by going to **Data > Describe data > Describe data contents (codebook)**
- If you click the ***Options*** tab, you will see that the dialog boxes are *persistent*—they remember their last state

---

Stata's Dialog Boxes: Tabs

- ***Main*** for the guts of the command.
- ***if/in*** for restricting the command to a subset of the dataset.
  Some commands have a ***by/if/in*** tab, which can be used for both subsets and for running the command on each subgroup in a dataset. We'll talk more about this when we get to the `by` prefix command.
  - For the curious: the ***if/in*** names come from the `if` and `in` qualifiers in Stata's command syntax. More later...
- ***Options*** for the options to the command
  - These are tweaks to the command which are necessarily command-specific
- There are often command-specific tabs—there is one for `codebook`: ***Languages*** tab.
  - Stata has multiple-language support for labels.

Choosing Variables

- Variables can be specified either by typing or by choosing from the list
- Multiple variables may be specified by clicking on multiple variables while the list is open
- Try choosing the `headroom` and `length` variables, and clicking **Submit**
  - Clicking **Submit** will leave the dialog box open
- Note: The drop boxes are nice for small datasets, but bad for large datasets—typing variable names is quicker in this case

---

Stata's Dialog Boxes: Main Buttons

- **OK**: submits the command to Stata and closes the dialog box
- **Cancel**: closes the dialog box without submitting a command
- **Submit**: submits the command and leaves the dialog box open
  - This is the right choice when experimenting

Stata's Dialog Boxes: Small Buttons

- **Help:** 
  - Opens help for the command
- **Reset:** 
  - Resets the dialog box to its default state, which typically is empty
- **Copy:** 
  - Copies the command to the clipboard
    - This is useful in automation, especially when using options

---

Trying the Help Button

- A Viewer window appears which explains the command syntax and explains the options.
- The sections which look like folder tabs explain each of the non-standard tabs in the particular dialog box.

Working with Commands

- The `codebook` command is simple enough that it is easy to type in the Command window.
- Try running `codebook` on `headroom` and `length`
- **Tip:** Variable names can be completed using the **Tab** in the Command window.
  - Try `codebook h` **Tab** `l` **Tab**
  - Quite a finger and brain-saver!

Hybrid Command and Dialog Box Method

- If you know the name of the command you want, you can type `db` *command name* to get the dialog box
- This is often much faster than navigating to the menu item
  - Note: this works most of the time—sometimes you may not get the dialog box, usually because of a multiword command name
- Try it now:
  - Close the `codebook` dialog
  - Type `db codebook` in the Command window and press **Enter**

Aside: What if Stata Doesn't Understand?

- If Stata can't understand what was typed in the Command window, it issues an error.
  - Errors are displayed in red.
  - Errors have error messages which explain why Stata is confused.
  - Errors have a return code, namely the number contained in the parentheses: `r(#)` (#, called "hash" is the number sign)
    - For more information click on the blue text!
- Check carefully for a typo - this is the usual problem.

Aside: Adding Comments

- If you would like to add a comment to your log file, type a command which starts with an asterisk (`*`)
- Anything following the asterisk will be ignored:

```
. * annotation helps recall the thought process
```

- Tip: It can be useful to start comments with something that won't show up naturally in your session
  - This makes searching the logs for comments easier
- An example which is used within Stata: double-bang (`!!`) for things needing attention

---

1.3.4 Looking Carefully at the Structure of a Dataset

Stata's Data Model—Data

- Stata works with a *one* dataset at a time
  - If you want to look at one dataset while working with another, open another instance of Stata
    - Note: you cannot log from the second instance into the log file of the first instance—all instances have separate log files!
- This is done for speed

Stata's Data Model—Datasets

- A dataset is made of a single table of data, along with its metadata
- We've brushed across three types of metadata already
  - *Labels* which describe the dataset, the variables, and values of encoded variables
  - *Notes* which are, well, free-form notes
  - *Formatting* which controls how data are displayed
- There are other kinds of metadata we've not looked at, yet
  - *Sort order*—important for reproducibility in bootstrapping

---

Aside: Advanced Metadata

- Stata tries to separate data structure from data analysis when possible
- Users can tell Stata that data have a special structure for various types of datasets
  - Time-series datasets—see `help tsset`
  - Survival data datasets—see `help stset`
  - Datasets arising from complex survey designs (called *survey* datasets)—see `help svyset`
  - Repeated measures/Panel data/Longitudinal data datasets—see `xtset`
- We'll work with less structured datasets for now

Looking at the Structure of a Dataset

- Look at **Data > Describe Data > Describe data in memory** to see the structure of the dataset in use
  - A very simple dialog box—click **OK**
- From the Results window, we see that the command is `describe`

```
Contains data from /Applications/Stata/ado/base/a/auto.dta
  obs:            74                          1978 Automobile Data
 vars:            12                          13 Apr 2007 17:45
 size:         3,478 (99.9% of memory free)    (_dta has notes)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
make            str18  %-18s                  Make and Model
price           int    %8.0gc                 Price
mpg             int    %8.0g                  Mileage (mpg)
rep78           int    %8.0g                  Repair Record 1978
headroom        float  %6.1f                  Headroom (in.)
```

```
trunk           int     %8.0g                 Trunk space (cu. ft.)
weight          int     %8.0gc                Weight (lbs.)
length          int     %8.0g                 Length (in.)
turn            int     %8.0g                 Turn Circle (ft.)
displacement    int     %8.0g                 Displacement (cu. in.)
gear_ratio      float   %6.2f                  Gear Ratio
foreign         byte    %8.0g       origin    Car type
------------------------------------------------------------------------------
Sorted by:  foreign
```

- This gives all the details about the structure of a dataset.

Variable Names

- Must start with a letter or underscore (_)
  - Avoid starting underscores, because they have a special meaning as temporary variables in a program.
- Then can be letters, underscores, or numbers.
- Upper and lowercase matter.
  - `make`, `Make`, and `MAKE` are all different names.
- Can **not** contain spaces!
- Have a maximum length of 32 characters

Details about Variable Types

- Stata has `str#` string variables for holding text.
- Stata distinguishes between various types of numbers, primarily for speed and efficiency.
  - See `help data types`
  - We'll look at these more when talking about data management.

Details about Display Formats

- Stata has a large number of ways to display numbers.
- There are so many, it is best to look at `help format`

Three Types of Labels

- The data label gives a brief description of the dataset. It is echoed when the dataset is opened.
- The variable labels give brief descriptions of the variables.
  - These are used in graphs and tables.
  - Should be self-explanatory.
- The value labels tell how to display encoded variables.
  - Makes numerical encodings human-readable.
  - The value labels are also used in graphs and tables.
- All labels can be multilingual.

How Does Stata Use a Dataset?

- Stata reads the dataset into memory
  - This is done for speed
- Stata does not touch the file on the disk, unless explicity told to do so
  - It is just like working on a word-processing document—nothing is saved until it is explicitly saved.
- If a dataset is too large to fit into memory, Stata needs to be told to grab more memory—we'll do this at the end of this lesson

---

1.3.5 Interface: Saving Time and Avoiding Carpal Tunnel

Reusing Commands

- Single commands may be recalled and edited by
  - Pressing **Page Up**
  - Single-clicking a command in the Review window
- Single commands can be rerun by double-clicking the command in the Review window.
- Multiple commands may be rerun by
  Note! These work in Stata 10, but not in Stata 9!
  - selecting them in the the Review window,
  - right-clicking one of the selected commands, and
  - selecting Do Selected...

Saving Typing for Variable Names

- Variable names may be completed in the Command window by pressing the **Tab** key.
  - If the name could expand to multiple variables, Stata will expand as far as it can.
  - This works only in the Command window.
- Variable names can be abbreviated in dialog boxes and the Command window
  - Useful in dialog boxes **but** this should be avoided when trying to replicable, just in case the variable names change
  - For an abbreviation to work, it must uniquely identify one variable.
- Stata also has wildcards for specifying variable lists.
  - More about this when discussing syntax.

1.4 Conclusion and Final Notes

1.4.1 Final Notes

Looking Back at the Log

- After working for a while, results will scroll off the top of the Results window
  - The size of the log file is fixed
- This is not a problem—look at the log file!
  - The log file can be opened in a Viewer window:
  - Click the **Log** button
  - Select the **View snapshot of log file** radio button
  - Click **OK**
- Clicking the refresh button will update the view when there is more in the log file

---

Changing Memory Allocation

- Before changing the memory allocation, Stata's memory must be cleared out completely

```
. clear all
```

- After this is done, the memory can be set—here we ask for 50MB of memory

```
. set memory 50m
```

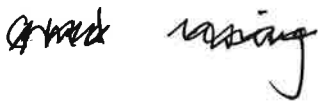
```
Current memory allocation
-------------------------

                    current                                    memory usage
    settable          value     description                    (1M = 1024k)
    ------------------------------------------------------------------------
    set maxvar         5000     max. variables allowed             1.751M
    set memory          50M     max. data space                   50.000M
    set matsize         400     max. RHS vars in models            1.254M
                                                                  ----------
                                                                   53.006M
```

- You can set memory to roughly 80% of your RAM
  - If you use Windows XP, this can be smaller, because of a bug in the Windows XP memory manager
  - Stata has a FAQ about this bug
    http://www.stata.com/support/faqs/win/winmemory.html

1.4.2 Conclusion

Finishing Up

- Close your log file
  - Click the **Log** button
  - Select the **Close log file** radio button
  - Click **OK**
- We could clear the memory, as above, but we don't really need to...



Lesson 2

Summaries and Tables

2.1 Introduction

2.1.1 Goals

Goals

- Learn some simple commands for summarizing data
- Reusing results
- Making tables
- Simple Tests

2.1.2 Getting Started

Starting Off

- Make a new folder to work in, say `table`.
- Start Stata.
- Open a log file.
- We'll use a dataset available over the web from Stata
  - In the Command window, type

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

    and press **Enter**
- Investigate the dataset

2.2 Basic Stats

2.2.1 Summary Statistics

Simple Summary Statistics

- Try **Statistics > Summaries, tables, and tests > Summary and descriptive statistics > Summary statistics**
- Click the **Submit** button
- A simple table of summary statistics appears

```
. summarize
```

Variable	Obs	Mean	Std. Dev.	Min	Max
id	189	121.0794	63.30363	4	226
low	189	.3121693	.4646093	0	1
age	189	23.2381	5.298678	14	45
lwt	189	129.8201	30.57515	80	250
race	189	1.846561	.9183422	1	3
smoke	189	.3915344	.4893898	0	1
ptl	189	.1957672	.4933419	0	3
ht	189	.0634921	.2444936	0	1
ui	189	.1481481	.3561903	0	1
ftv	189	.7936508	1.059286	0	6
bwt	189	2944.286	729.016	709	4990

- We see that the `summarize` command is used to get simple summary statistics
  - This is simpler to type then to select!

Variations on a Theme

- Try changing the variable to `age`, and checking the **Display additional statistics** radio button

```
. summarize age, detail
```

age of mother

Percentiles		Smallest		
1%	14	14		
5%	16	14		
10%	17	14	Obs	189
25%	19	15	Sum of Wgt.	189
50%	23		Mean	23.2381
		Largest	Std. Dev.	5.298678
75%	26	35		
90%	31	36	Variance	28.07599
95%	32	36	Skewness	.7164391
99%	36	45	Kurtosis	3.568442

- More statistics about shape and percentiles appears
- Try experimenting a little, here
- You can see that checking the different options results in a command with a comma, and some additional instruction

2.2.2 Reusing Results

Returned Results

- Try summarizing the birthweights (`bwt`) with details

```
. summarize bwt, detail
```

```
                    birthweight (grams)
-------------------------------------------------------------
      Percentiles      Smallest
 1%         1021            709
 5%         1790           1021
10%         1970           1135       Obs                 189
25%         2414           1330       Sum of Wgt.         189

50%         2977                      Mean           2944.286
                        Largest       Std. Dev.       729.016
75%         3475           4174
90%         3884           4238       Variance       531464.4
95%         3997           4593       Skewness      -.2069782
99%         4593           4990       Kurtosis       2.888821
```

- To see what Stata has stored look at this:

```
. return list
```

```
scalars:
                  r(N) =  189
              r(sum_w) =  189
               r(mean) =  2944.285714285714
                r(Var) =  531464.3541033434
                 r(sd) =  729.0160177275554
           r(skewness) =  -.2069781935638592
           r(kurtosis) =  2.888821334996583
                r(sum) =  556470
                r(min) =  709
                r(max) =  4990
                 r(p1) =  1021
                 r(p5) =  1790
                r(p10) =  1970
                r(p25) =  2414
                r(p50) =  2977
                r(p75) =  3475
                r(p90) =  3884
                r(p95) =  3997
                r(p99) =  4593
```

Scalar range = r(max) − r(min)

- Stata lists some *scalars* with values corresponding to the last variable summarized (here: `bwt`)
- This can be used in computations
  - The range could be computed:

```
. display r(max) - r(min)
4281
```

Aside: Using `display` for computations

- The `display` command displays results to the Results window
- You can do computations using display
- Example: changing the mean birthweight from grams to pounds:

```
. display r(mean)/453.6
6.4909297
```

What is with the `r()`?

- You may have noticed that all the returned results looked like `r(`*something*`)`
- These are all the "returned" results
  - Stata has a name for these: these are *r-class* results
  - The technophilic name is used, because there can also be estimation results (e-class) and system (c-class)

Notes on Returned Results

- The returned results will be available until the next command which returns any r-class results..., which is almost every command other than `display`
- Sometimes *macros* will be returned—these hold text rather than numbers.
  - They can be displayed using `display "`r(`whatever`)'"`
  - These are becoming increasingly rare

2.2.3 Tables

Stata and Tables

- Stata has a variety of ways to make tables: look at **Statistics > Summaries, tables, and tests > Tables**
- Let's learn a bit about tables from the dialog boxes

One-way Tables

- A one-way table simply summarizes either counts or another variable across the values (levels) of one variable (factor)
- Go to **Statistics > Summaries, tables, and tests > Tables > One-way tables**
- Make a table of the values of race
- The command is simple

```
. tabulate race
```

race	Freq.	Percent	Cum.
white	96	50.79	50.79
black	26	13.76	64.55
other	67	35.45	100.00
Total	189	100.00	

Two-way Tables

- A two-way table either counts frequencies or summarizes another variable across the values of two variables
- Go to **Statistics > Summaries, tables, and tests > Tables > Two-way tables with measures of association**
- Make a table of the values of race *vs.* low birthweight
- The command is simple, again:

```
. tabulate race low
```

race	birthweight<2500g 0	1	Total
white	73	23	96
black	15	11	26
other	42	25	67
Total	130	59	189

- Try some of the hypothesis tests

Summary Statistics within Tables

- It is possible to get summary statistics within one- and two-way tables
- Go to **Statistics > Summaries, tables, and tests > Tables > One/two-way table of summary statistics**
- Make a table summarizing the birthweights across race and smoking status
- The command is still simple

```
. tabulate race smoke, summarize(bwt)
```

```
Means, Standard Deviations and Frequencies of birthweight (grams)

           |    smoked during
           |      pregnancy
      race |         0          1 |     Total
-----------+----------------------+----------
     white |   3428.75  2827.3846 | 3103.0104
           | 710.09892  626.68443 | 727.87244
           |        44         52 |        96
-----------+----------------------+----------
     black |    2854.5       2504 | 2719.6923
           | 621.25432  637.05677 | 638.68388
           |        16         10 |        26
-----------+----------------------+----------
     other | 2814.2364  2757.1667 | 2804.0149
           |  708.2607  810.04465 | 721.30115
           |        55         12 |        67
-----------+----------------------+----------
     Total | 3054.9565  2772.2973 | 2944.2857
           | 752.40901  659.80748 | 729.01602
           |       115         74 |       189
```

Aside: A Note on the `tabulate` Command

- The `tabulate` command has many uses
- It can be used to generate indicator (dummy) variables
- Not-quite-subliminal message: we should learn how commands work directly...

More One-way Tables of Statistics

- For making tables summary statistics, go to **Statistics > Summaries, tables, and tests > Tables > Table of summary statistics**
- This brings up a dialog box for the `tabstat` command
  - Try selecting `age`, `lwt`, and `bwt`

  - Choose *mean*, *Standard deviation*, *median*, and *Interquartile range* as statistics
  - Click the ***Options*** tab, and select *Statistics* from the *Use as columns:* combo box
  - Click OK

- A more complex command

```
. tabstat bwt age lwt, ///
.   statistics( mean sd median iqr ) ///
.   columns(statistics)

    variable |      mean        sd       p50       iqr
-------------+--------------------------------------------
         bwt |  2944.286   729.016      2977      1061
         age |   23.2381  5.298678        23         7
         lwt |  129.8201  30.57515       121        30
----------------------------------------------------------
```

More Complex Tables

- `table` will allow multiway tables of summary statistics
- Other tables for epidemiologists are available under **Statistics > Epidemiology and related > Tables for epidemiologists**
  - To see help, try `help epitab`

2.2.4 Subgroups

Subgroups

- Suppose we would like to make tables by various subgroups
  - This is another way to make three-way tables
- We need to tell Stata how to define the subgroups
- It can then create a table for each subgroup
- Stata will run a command repeatedly on subgroups if an `by` prefix command is used

---

Subgroups via Dialogs

- Go to **Statistics > Summaries, tables, and tests > Tables > One/two-way table of summary statistics**, again
- On the ***Main*** tab,
  - Set *Variable 1* to `ui`, *Variable 2* to `smoke`, and *Summarize variable* to `bwt`
- Now click the ***by/if/in*** tab
  - Check the **Repeat command by groups** checkbox, and put `race` in for the variable
- Click **OK**

```
. by race, sort : tabulate ui smoke ///
.   , summarize(bwt)

-------------------------------------------------------------------------------
-> race = white

     Means, Standard Deviations and Frequencies of birthweight (grams)

 presence, |
   uterine |    smoked during
irritabili |      pregnancy
        ty |         0          1 |     Total
-----------+----------------------+----------
         0 |    3494.5  2874.1628 | 3173.1205
           | 622.51763  634.65157 | 698.47409
           |        40         43 |        83
-----------+----------------------+----------
         1 |   2771.25  2603.8889 | 2655.3846
           | 1247.2093  566.66645 | 780.65384
           |         4          9 |        13
-----------+----------------------+----------
     Total |   3428.75  2827.3846 | 3103.0104
           | 710.09892  626.68443 | 727.87244
           |        44         52 |        96

-------------------------------------------------------------------------------
-> race = black

     Means, Standard Deviations and Frequencies of birthweight (grams)

 presence, |
   uterine |    smoked during
irritabili |      pregnancy
        ty |         0          1 |     Total
-----------+----------------------+----------
         0 | 3002.6154       2504 | 2785.8261
           | 583.83125  637.05677 | 644.84344
           |        13         10 |        23
-----------+----------------------+----------
         1 | 2212.6667          . | 2212.6667
           | 298.32924          . | 298.32924
```

```
           |         3          0 |         3
-----------+----------------------+----------
     Total |    2854.5       2504 | 2719.6923
           | 621.25432  637.05677 | 638.68388
           |        16         10 |        26

-------------------------------------------------------------------------------
-> race = other

    Means, Standard Deviations and Frequencies of birthweight (grams)

 presence, |
   uterine |    smoked during
irritabili |      pregnancy
        ty |         0          1 |     Total
-----------+----------------------+----------
         0 |  2884.617    3104.75 | 2916.6364
           | 700.49453  407.02676 |  667.5388
           |        47          8 |        55
-----------+----------------------+----------
         1 |   2400.75       2062 | 2287.8333
           | 645.39639  1026.1027 | 761.60236
           |         8          4 |        12
-----------+----------------------+----------
     Total | 2814.2364  2757.1667 | 2804.0149
           |  708.2607  810.04465 | 721.30115
           |        55         12 |        67
```

The by Prefix Command

- You can see that the following odd-looking prefix was used:

  `by race, sort:`

- This tells Stata to use the `race` variable to split up the dataset, and then run the `tabulate` command on each subgroup
  - The `sort` option is needed, because it changes sort order
  - An alternative to `by... , sort:` is `bysort... :`
- We'll discuss this in more detail, later, when we know more about Stata commands

2.3 Conclusion and Finishing Up

2.3.1 Conclusion

Finishing Up

- Close your log file
  - If you'd like to do this from the Command window, type

```
log close
```

Table Thoughts

- For complex tables, it looks like the dialog boxes are useful
- For simple tables, it looks like learning the `tabulate` command is worthwhile to learn
- We need a way to do this!

What Did We See?

- It is simple enough to get summary statistics and some tables using the menus and dialogs
- The menus and dialogs can be slow for simple commands
- The commands for the tables and summary statistics are not very complex in general

---

Lesson 3

Stata's Syntax

3.1 Introduction

3.1.1 Goals

Goals

- Learning All about Syntax
- Understanding Clauses
- Understanding Expressions

3.1.2 Getting Started

Starting Off

- Make a new folder to work in, say `syntax`
- Start Stata
- Open a log file called `syntax`
- Open the `auto` dataset which comes with Stata

```
. sysuse auto
```

3.2 Stata's Syntax

3.2.1 Basics

Strength through Simplicity

- Stata has a very uniform syntax for its commands.
  - This is something different for statistics packages!
- Learning the most basic syntax will allow using most commands quickly.
- Both the manuals and the on-line help assume that you understand syntax diagrams.
  - Commands can typically be run using a dialog box, but the explanations use Stata commands.

Strength through Consistency

- Stata's estimation commands are consistent within type.
  - If you know the syntax for a linear regression, you know the syntax for a logistic regression.
- Stata's post-estimation commands are consistent across models.
  - The post-estimation commands are used in a similar fashion across all applicable models.
- This means that learning a little is learning a lot!

Stata's Typical Command Syntax

- There first general form for a Stata command is:

  `command` [*varlist*] [*if*] [*in*] [[*weight*]] [`,` *options*]
- The chunk of a command before the comma is called the *standard* part.
- The chunk after the comma is the *options* part.
- Think of them as the "common" and "specific" parts.

Some Other Common Syntaxes

- There are two other common syntaxes, which are highly specific
- When creating or changing values of variables, there is an assignment clause

  `command` *varname* `=`*exp* [*if*] [*in*] [`weight`] [`, options`]
- When working with external files, there is often an extra `using "`*filename*`"`

Two Parts: Standard and Options

- The standard part is so called, because it contains clauses which appear in many commands.
  - Could also be called the common part of the command.
  - Not all commands allow all clauses.
  - Sometimes clauses are required, rather than optional.
- The options part are items which are specific to the command in question.
  - Could also be called the specific part of the command.
  - The options differ greatly from command to command.
  - Options are sometimes required.
    - Don't confuse option with optional!

The Full Picture

- To see the full picture of how Stata's syntax works, type

  `help language`

  Typing `help syntax` takes you to the Stata `syntax` command. The `syntax` command allows writing programs which have exactly the same standard syntax as Stata's own commands. We'll see a bit about this when talking about programming.

- This gives the details for all the pieces of Stata's commands.
- We'll go over some of them here.

Understanding the Clauses

- The trick to being able to piece together the syntax of a Stata command is to understand
  - The clauses in the standard part (not too bad)
  - The phrases used within the clauses (needs explanation)
  - The options (command-specific)
- The starting point here will be to look at the help, to see some examples of options.

Help for `codebook`

- To see the help for the `codebook` command, type `help codebook`
- Since everything but `codebook` is in square brackets, the minimum command is `codebook`
- `codebook` can be used with 0 or more variables.
  - When the *varlist* is optional, leaving it out uses all variables.
- The (still mysterious) `if` and `in` qualifiers are allowed.

Help for `describe`

- To get help from the Viewer directly, type the name of the command in the address bar—try this for `describe`
- We see old things:
  - The *varlist* is optional.
- We see a few new things:
  - The `d` in `describe` command is underlined
  - Two syntaxes, one for `using` with an external file, one for using the data in memory.
    - We used the latter earlier.

Command Abbreviations

- The underlined portion of a command or option is the minimum allowable abbreviation
  - So, for `describe`, the minimum possible command is `d`
  - Typically, the more often a command would be used, the shorter the abbreviation
  - Clearly, this is considered an oft-used command!
- A note about using command abbreviations: *What helps typing hurts readability!* Use abbreviations only when they don't affect readability.
  - Except for `d`, of course.

Checking syntax for `summarize`

- Try getting help for the `summarize` command.
- Answer the following to yourself:
  - What is required?
  - What is optional?
  - What is the minimum abbreviation of the command?

Abbreviations for Known Commands

- Try looking abbreviations for other commands you've seen:
  - `su` for summarize (though `sum` is more readable)
  - `tab` for tabulate

3.2.2 Variable lists

What is a *varlist*?

- A *varlist* is a list of one or more variables.
  - That was simple!
- There are many ways of abbreviating the names of variables, though.

Abbreviated Variable Names

- Stata will understand an abbreviated variable name if only one variable starts with that abbreviation
  - This can be dangerous!
- Examples:
  - `headroom` could be abbreviated to `h`, since there is no other variable whose name starts with `h`.
  - `displacement` can be abbreviated as `d`
  - Try it: `d d`
- Antiexample:
  - `m` would not work as an abbreviation, because both `mpg` and `make` start with `m`.
  - Try it (and get an error): `sum m`

Wildcards

- Stata understands three wildcards
  - The * stands for 0 or more characters
    - Can be placed anywhere in the name
    - Useful for groups of similar variables
  - The ? stands for exactly 1 character.
    - Not so useful
  - The ~ acts just like the *, except
    - It is valid only if its expansion results in a unique variable name
    - It can not be used to start a variable name
    - Not of much use at all
  - Multiple wildcards can be used in one variable name.

Examples of *

- Examples of *:
  - `m*` stands for anything starting with `m`: `mpg` and `make`
  - `*t` stands for anything ending in `t`: `weight` and `displacement`
- Try these with the summarize command:

```
. sum m* *t

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
        make |         0
         mpg |        74     21.2973    5.785503        12         41
      weight |        74     3019.459    777.1936      1760       4840
displacement |        74     197.2973    91.83722        79        425
```

Variable Ranges

- Stata also understands variable ranges:
  - *firstvar-lastvar*
  - expands to all variables from firstvar to lastvar
    - lastvar must follow firstvar in the dataset
- Example:
  - `head-disp` stands for every variable from `headroom` to `displacement`

```
. sum head-disp

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
    headroom |        74    2.993243    .8459948       1.5          5
       trunk |        74    13.75676    4.277404         5         23
      weight |        74    3019.459    777.1936      1760       4840
      length |        74    187.9324    22.26634       142        233
        turn |        74    39.64865    4.399354        31         51
-------------+--------------------------------------------------------
displacement |        74    197.2973    91.83722        79        425
```

  - `disp-head` gives an error because `headroom` is stored earlier in the dataset.

- Good for extracting large numbers of variables, but not too useful for typical daily usage.

sum _all (for all variables)

Why to Use and Avoid Abbreviations

- Why to Use
  - Wildcards are useful when working with a group similar variables
  - Abbreviations can be useful in dialog boxes
- Why to Avoid
  - While abbreviating commands affects only readability, abbreviating variable names can affect whether commands work after changes to a dataset
    - Imagine adding a `dealer` variable—suddenly `d d` would fail!
  - Tab completion works as well in the Command window, and keeps full names

---

Notes About the *varlist* Clause

- If a *varlist* is optional for a command, not specifying any variables will (generally) cause the command to run on all the variables
  - We've seen this in the `describe`, `codebook` and `summarize` commands
- To specify all variables explicitly, `_all` can be specified. The underscore is intentional!
- If a command refers to a *varname* rather than `varlist`, then only one variable may be used

---

3.2.3 Working with a Subset of the Data—And Evaluating Expressions

The `if` and `in` Qualifiers

- These two qualifiers are used to use a partial dataset for a command
- They are used command by command—there is no way to tell Stata "Use this subset of the data until further notice"
  - The only way to do this is to keep just the subset of interest, and then to reload the dataset later
  - This is a good thing—it is too easy to forget that only a subset of the data are in use

---

The `in` Qualifier

- The `in` qualifier allows subsetting by observation number
  - `in` # looks at observation #
  - `in` #/# looks at observations # through #
    - Negative numbers count from the end of the dataset, so `-1` is the last observation
    - The letters `f` and `l` can be used for first and last
- The `in` is most useful when listing output or browsing
  - Examples:
    - Listing `make` and `mpg` for the first 5 observations

```
. list make mpg in 1/5
     +--------------------+
     | make            mpg |
     |--------------------|
  1. | AMC Concord      22 |
  2. | AMC Pacer        17 |
  3. | AMC Spirit       22 |
  4. | Buick Century    20 |
  5. | Buick Electra    15 |
     +--------------------+
```

    - Listing the same for the last 8 observations

```
. list make mpg in -8/-1
     +----------------------+
     | make             mpg |
     |----------------------|
 67. | Toyota Celica     18 |
 68. | Toyota Corolla    31 |
 69. | Toyota Corona     18 |
 70. | VW Dasher         23 |
 71. | VW Diesel         41 |
     |----------------------|
 72. | VW Rabbit         25 |
 73. | VW Scirocco       25 |
 74. | Volvo 260         17 |
     +----------------------+
```

- Mostly of use when exploring, or when the sort order is meaningful

---

The `if` Qualifier

- The `if` qualifier allows subsetting by evaluating an expression
  - `if` *exp*
- It's really quite simple:
  - Stata evaluates the expression in the `if` qualifier for each observation.
  - If the result is "true", then the observation is used for the command.
  - If the result is "false", then the observation is excluded from the command.
- In this context, the expression is typically a logical expression.

Examples of `if`

- Simple Example
  - `browse make mpg foreign if mpg<20`
    - Shows just those observations for which the mpg is less than 20.
- Here is something surprising
  - `browse make price rep78 if rep78 >= 4`
  - Yow! The observations with missing values are included!
    - Remember that missing numerical values are infinity-like.
  - How could this be fixed?
    - Would need to know how to specify missing values and what to use for "and".

Expressions

- To understand how to build something more complex, we need to know what *expressions* are
- An expression is something that can be evaluated and which returns a value
  - `2 + 2` is an expression, though not interesting
  - `1/mpg` is an expression which does some computation
  - `mpg < 20` looks logical, but as we see, it produces 0's and 1's
- Expressions are built from *constants*, *variables*, *operators*, and *functions*

Operators

- Operators are symbols which manipulate values
  - If two values are needed, then they must be of the same type
    - Cannot add strings to numbers, for example
  - Spaces may be used on either side of operators for readability, but are not required
  - The values operated on by the operators are called operands
- The types of operators are
  - String
  - Arithmetic
  - Relational
  - Logical

First, and Least: String Operators

- These operators can work only with strings (text).
- Stata has just one string operator
  - Concatenation: `+`
- This also works with both variables and literals (i.e. bits of text not in a variable):
  - Examples:
    - The result of `"my" + "dog"` is `"mydog"`
    - Assuming that `fname` and `lname` are string variables, the result of `fname + " " + lname` is the contents of `fname`, a space, and the contents of `lname` for every observation in the dataset.

Arithmetic Operators

- Stata uses the 5 standard arithmetic operators:
  - Addition: `+`
  - Subtraction: `-`
  - Multiplication: `*`
  - Division: `/`
  - Exponentiation: `^`
- The operands can be either numbers or numeric variables.
- If either value is missing, the result will be missing.

Relational Operators

- Relational operators compare size or order, and have the (abstract) result of true or false.
- Stata uses these:
  - At least (greater than or equal to): `>=`
  - Greater than: `>`
  - Equal to: `==`
    - Note the double equal sign—"really equal"
    - The single equal signs is used for assignment in expressions.
  - Less than: `<`
  - At most (less than or equal to): `<=`
  - Not equal: `!=`

Logical Operators

- Logical operators (abstractly)
  - have logical operands,
  - evaluate to either "True" or "False"

Logical Operators II

- The logical operators are
  - And: `&`
    - *A* `&` *B* is true if and only if both *A* and *B* are true.
  - Or: `|` (found just above the Enter key)
    - *A* `|` *B* is true if *A* or *B* (or both) is true, and false if and only if both *A* and *B* are false
  - Not: `!`
    - `!`*A* is false if *A* is true, and true if *A* is false.
    - This is not used as commonly as the conjunctions - it's main use comes in making complicated expressions easier to read. Most times `!=` is what is needed, rather than the plain `!`

Example of Logical and Relational Operators

Stata consider . as a bigger value.

- Fixing the problem of the large missing values.
  - `browse make price rep78 if rep78 >= 4 & rep78 < .`
- Better: Stata has a `missing()` function for checking missing values without looking obscure. Using this, the expression becomes

```
browse make price rep78 if rep78 >= 4 & !missing(rep78)
```

- There are many operators in the above expression—how can we be sure it evaluates correctly?
  - Use parentheses for grouping
  - Memorize many levels of precedence–ugh

Another Example

- Suppose a car is classified as a "tank" if it gets less than 18 miles to the gallon or is 200 inches long or longer
  - `browse make mpg length if ((mpg < 18) | (length >=200)) & !missing(length)`

Stata and Logic

- Typically think of logic as working with "True" and "False"
- Stata has no boolean data type, though, so for inputs to a logical expression
  - Zero is considered "False"
  - Anything non-zero is considered "True", even missing values(!)
- The result of a logical expression is 1 for "True", and 0 for "False".
  - So... the expression 1 > 0 has the value 1.

Exploiting This Logic

- If a variable is an indicator with no missing values, it can be used directly

```
. sum mpg if foreign

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
         mpg |        22    24.77273    6.611187         14         41
```

  - Such commands are very readable—but care is needed for missing values
- We can use this to make new indicator variables easily (see below)

3.2.4 Weights

`weights`

- Weights are used in a variety of situations, each of which corresponds to a situation where the different observations should not be treated as equal when doing computations:
  - When each observation represents a group of people.
    - Example: each observation represents an entire state, and an average is needed.
  - When complicated sampling schemes cause different groups of people to have different chances of being sampled.
    - Those who can be sampled more easily contribute less information.

The weight Clause Details

- The `[weight]` in the syntax is replaced by a phrase of the form
  [*weightword* = *expression*]
  - The square brackets are typed.
  - The *weightword*s are outlined on the following pages.

---

weightwords Part 1

- `fweight` or `frequency` for frequencies
  - The expression represents frequencies
- `pweight` for probability weights
  - The expression represents the inverse of the probability of being sampled.
    - i.e. (1/probability of being sampled)
- `aweight` for analytic weights
  - The expression is are inversely proportional to the variance of the value of a variable.

---

weightwords Part 2

- `iweight` for "importance" weights
  - No real meaning, but used to implement weights which do not have a concrete interpretation such as the above.
- `weight` for the default weights for the particular command
  - Not good for documentation. It is better to specify exactly that type weights which is desired.

Notes on `weight`

- The weight clause is one of the four exceptions where the square brackets are typed - they are not just part of the syntax diagram!
- We will see examples of weights shortly

3.2.5 Other Standard Pieces

The Assignment Clause

- An assignment clause tells Stata to put the results of the right-hand side (RHS) of the equal sign into whatever is on the left-hand side (LHS) of the equal sign
  - The equal sign is called the assignment operator in this case, because it assigns the RHS to the LHS
- Here's an example which uses the (new) `generate` command:

```
. generate price_per_lb = price/weight
```

  - For each observation, the new variable `price_per_lb` is filled with the ratio of price and the weight.
    - If `price` or `weight` would be missing, so would `price_per_lb`
  - We can exploit Stata's logic to mark all the gas hogs

```
. generate gashog = mpg < 20
```

Functions

- Stata has a large number of functions type `help functions` to see the help.
- Functions have the form *functionname*`(`*arguments*`)`
  - There can be *no space* between the function name and the left parenthesis.
  - The arguments are the values which the function will use. They themselves can be expressions.
- Example: the `log10()` function finds logarithms base 10.

The `using` Clause

- This construct is for accessing information from data files which are not currently in use.
  - This will be used when importing data into Stata.
- The file name can be specified either by
  - Typing it in within quotation marks (often hard)
  - Letting the computer specify it (easy)
    - Go to **File > Filename...**
- We'll see the `using` clause when putting data sets together using the `append` and `merge` commands.

Phew! A Brief Review.

- The assignment clause is used to assign values.
- The `if` qualifier is used to run a command on a subset of the dataset.
- Both of these use expressions.
- Expressions are built from numbers, strings, variables, operators, and functions.
- Stata employs 0s and 1s for logic.
- Any place there is an *exp* in a Stata syntax, a general expression may be substituted.

Common Syntax Errors

- Misspelling a command name or a variable name
- Separating different options with commas.
  - Only one comma is needed to separate the standard part of the command from the options.
- Using a single = sign when testing for equality
  - Stata tests equality with ==
- Putting a space between a function or option name and the left parenthesis:
  - Good: `log10(income)`
  - Bad: `log10 (income)`

Common Ways to Fix Syntax Errors

- Read the error message which Stata gives. Many times this will give you a clue.
  - Many times it won't, also, because it is trying to find a mistake in something it does not understand.
- Check for a simple typo.
- Check the syntax of the command (using help).
- Build up a complicated command piece by piece, using the dialogs, then compare.

3.3 Conclusion and Final Notes

3.3.1 Final Notes

Notes on Missing Values

- When comparing numbers, remember that missing values are the largest possible value!
  - Missing values do not cause missing comparisons!

Notes on Comparing Numbers

- Some care is needed when comparing non-integer variable's values to non-integer numbers
  - In the `auto` dataset, many observations have a gear ratio of 2.93
  - Try listing them

```
. list if gear_ratio == 2.93
```

  - Nothing appears!
- What is the problem?
  - `gear_ratio` is stored as a float (single-precision number)
  - `2.93` is a number, and hence a double-precision number

Floats *vs.* Doubles

- Variables stored as floats have less precision than real numbers, which are treated as doubles.
- Floats carry fewer digits than doubles, so they can be very slightly different in the ending digits. Try the following:

```
. display %20.18f 2.93

2.930000000000000160
```

- The result is different than if the number is treated as a float:

```
. display %20.18f float(2.93)

2.930000006675720148
```

- Lesson: If comparing a variable stored as a `float` to a non-integer, use the `float()` function:

```
. list if gear_ratio == float(2.93)
```

```
     +-------------------------------------------------------------------------+
4.   | make             |  price | mpg | rep78 | headroom | trunk | weight |
     | Buick Century    |  4,816 |  20 |     3 |      4.5 |    16 |  3,250 |
     |-------------------------------------------------------------------------|
     | length | turn | displa~t | gear_r~o |  foreign | price_~b | gashog   |
     |    196 |   40 |      196 |     2.93 | Domestic | 1.481846 |      0   |
     +-------------------------------------------------------------------------+

     +-------------------------------------------------------------------------+
8.   | make             |  price | mpg | rep78 | headroom | trunk | weight |
     | Buick Regal      |  5,189 |  20 |     3 |      2.0 |    16 |  3,280 |
     |-------------------------------------------------------------------------|
     | length | turn | displa~t | gear_r~o |  foreign | price_~b | gashog   |
     |    200 |   42 |      196 |     2.93 | Domestic | 1.582012 |      0   |
     +-------------------------------------------------------------------------+

     +-------------------------------------------------------------------------+
9.   | make             |  price | mpg | rep78 | headroom | trunk | weight |
     | Buick Riviera    | 10,372 |  16 |     3 |      3.5 |    17 |  3,880 |
     |-------------------------------------------------------------------------|
     | length | turn | displa~t | gear_r~o |  foreign | price_~b | gashog   |
     |    207 |   43 |      231 |     2.93 | Domestic | 2.673196 |      1   |
     +-------------------------------------------------------------------------+

     +-------------------------------------------------------------------------+
14.  | make             |  price | mpg | rep78 | headroom | trunk | weight |
     | Chev. Chevette   |  3,299 |  29 |     3 |      2.5 |     9 |  2,110 |
     |-------------------------------------------------------------------------|
     | length | turn | displa~t | gear_r~o |  foreign | price_~b | gashog   |
     |    163 |   34 |      231 |     2.93 | Domestic | 1.563507 |      0   |
     +-------------------------------------------------------------------------+

     +-------------------------------------------------------------------------+
36.  | make             |  price | mpg | rep78 | headroom | trunk | weight |
     | Olds Cutl Supr   |  5,172 |  19 |     3 |      2.0 |    16 |  3,310 |
     |-------------------------------------------------------------------------|
     | length | turn | displa~t | gear_r~o |  foreign | price_~b | gashog   |
     |    198 |   42 |      231 |     2.93 | Domestic | 1.562538 |      1   |
     +-------------------------------------------------------------------------+

     +-------------------------------------------------------------------------+
37.  | make             |  price | mpg | rep78 | headroom | trunk | weight |
     | Olds Cutlass     |  4,733 |  19 |     3 |      4.5 |    16 |  3,300 |
     |-------------------------------------------------------------------------|
     | length | turn | displa~t | gear_r~o |  foreign | price_~b | gashog   |
     |    198 |   42 |      231 |     2.93 | Domestic | 1.434242 |      1   |
     +-------------------------------------------------------------------------+

     +-------------------------------------------------------------------------+
49.  | make             |  price | mpg | rep78 | headroom | trunk | weight |
     | Pont. Grand Prix |  5,222 |  19 |     3 |      2.0 |    16 |  3,210 |
     |-------------------------------------------------------------------------|
     | length | turn | displa~t | gear_r~o |  foreign | price_~b | gashog   |
     |    201 |   45 |      231 |     2.93 | Domestic | 1.626791 |      1   |
     +-------------------------------------------------------------------------+
```

```
     +-------------------------------------------------------------------------+
50.  | make             |  price | mpg | rep78 | headroom | trunk | weight |
     | Pont. Le Mans    |  4,723 |  19 |     3 |      3.5 |    17 |  3,200 |
     |-------------------------------------------------------------------------|
     | length | turn | displa~t | gear_r~o |  foreign | price_~b | gashog  |
     |    199 |   40 |      231 |     2.93 | Domestic | 1.475937 |      1  |
     +-------------------------------------------------------------------------+
```

Notes on Comparing Strings

- When comparing strings, ASCII order is used
  - uppercase before lowercase
    - `"This"` is before (less than) `"that"`
  - alphabetical within case
    - `"this"` is after (greater than) `"that"`

3.3.2 Conclusion

Finishing Up

- Close the log file
- May as well clear out the dataset

```
. clear
```

Lesson 4

Using Help

4.1 Introduction

4.1.1 Goals

Goals

- Learning to use Help
- Using Stata's web-connectivity and the web
- Installing and uninstalling user-written packages

4.1.2 Starting Up

Usual Routine

- Make a folder for this topic, say `help`
- Change your working directory
- Start a log file called `help`

4.2 Help

4.2.1 Help

The `help` command

- By itself, typing `help` will open a new Viewer window with links to help to introductory topics.

- With an argument, the `help` command is used to get help on commands.
  - It actually finds properly named help files—some help files do not correspond to commands.
- If a command is not found, it will ask if it should run a search on the keywords included with Stata.

Examples

- `help`
  - Goes to the table of contents for the Help system.
- `help codebook`
  - Brings up the help for the `codebook` command.
- `help import`
  - Cannot find a function, and searches Stata's keyword files for `import`

How Useful?

- This is useful for known commands or help files.
  - `help language`
    for syntax, for example.
- It is also surprisingly useful for using dialog boxes.
  - Try looking at `help use`, and then clicking the two links in the upper right-hand corner.
- It is not as useful if the command or help file is not known.

4.2.2 Casting a Wider Net

The `search` Command

- Look it up in the Help system: `help search`
  - Using the `search` from the Command window puts the results in the Results window.
  - Using `search` in a Viewer window puts the results in the Viewer window.
- There are many options—all for setting the scope of the search.
- Scroll down to the Advice section
- Here is your best advice on using `help search`!

The `findit` Command

- This is equivalent to using `search ... , all`
- The simplest way to search everything.
- Try this: `findit dummy`

The `hsearch` Command

- Look at the bottom of the help under **Also see**—click the link.
- This is a full word-search of all the help files. The index takes some time to build, so it is better tried outside this lab.
- It will find anything internal to Stata's help files, but it is a bit more cryptic than the other

---

4.3 The Viewer

4.3.1 The Viewer as Browser

Viewer Windows

- When we looked for help, Stata opened what it calls a "Viewer" window
- The Viewer windows are meant to be web-browser like
  - Instead of reading web pages, they read text files and files written in smcl

---

Address Bar

- The top bar is like an address bar in a web browser, except that the addresses commands related to the Viewer.
  - Some commands, such as `search` can be typed in here.
  - Net-related commands (see below) can be typed in here.
  - Otherwise, typing *words* here is like typing `help` *words* in the Command window.
- Try getting help for `tabulate`.

The Search Bar

- The bar next to the will run the `search` command.
- The drop menu allows setting the scope.
  - Not as richly as the options to `search`, but containing the three most common options.
- Try searching here for `goodness of fit`

The Find Bar

- Clicking the binoculars (or pressing **Ctrl-F**) allows searching for text on the page.
- Each instance is highlighted.

Clicking Links

- All blue links are links, just like in a web browser.
- Hovering over the link brings up the address or command that would be issued if the link were clicked.
- Clicking links to http URLs will bring up the proper page in your browser.

Closing Excess Windows

- After often searching for topics or help, there can be many Viewers.
- Go to **Window > Viewer > Close all Viewers**

4.4 Stata's Web Connectivity

4.4.1 Using External Datasets

Getting Datasets from Stata

- Select **File > Example Datasets...**
- Click the `Stata 10 manual datasets` link.
- Click the `Base Reference Manual [R]` link.
- Search the page for *logistic* to find the files which go with the `logistic` command
- Click the `describe` link next to the `lbw` dataset to see the structure of the dataset

---

4.4.2 Extending Stata

Links to Files

- Some links on help pages lead to `package` files which allow files to be downloaded to your computer, for example

```
findit visualizing
```

- Try one—it won't install yet!
- You'll see that there are help files, which let you see if you would like to install the files.
- Clicking `click here to install` will install the files in such a fashion that you can use them as though they were shipped with Stata.
- Links which say `click here to get` will download the shown files to your current working directory.

---

help hsearch

A Simple Example of Installing a Package

- Search in the search bar for `xk8`
  This package has one purpose: illustrating installing a package. It does nothing interesting.
- Click on the link to the `xk8` package.
- Check the `xsample.hlp` help file.
- Click the back arrow.
- Click the `click here to install` link.
- Go the Command window and type `xsample`

Seeing What Packages You Have

- Typing `ado dir` will show your installed packages.
  - Typed in the Command window puts the results in the Results window (easier).
  - Typed in the Address bar of the Viewer puts the results in the Viewer.

Deleting Packages

- If you click a link to an installed package from the `ado dir` results, you can uninstall the package.
- You can also type `ado uninstall` *package identifier*
  - The package identifier is in bold after the word `package`
    - Here, the package identifier is `xsample`

4.4.3 Using the Viewer to Download Files

Ancillary Files

- Files which do not extend Stata's capabilities are called *ancillary files* by Stata.
- They are typically data files or other files needed for books which use Stata for teaching.
- These are good for sharing data files.
- We'll see this usage in later lessons.

4.5 Conclusion

4.5.1 Conclusion

Conclusion

- We now can use help to find command syntaxes.
- Combined with our knowledge of command syntax, this means that we should be able to read and type commands.

Finishing Up

- Close your log file

Lesson 5

Tests and CIs

5.1 Introduction

5.1.1 Goals

Goals

- Making confidence intervals
- Running simple hypothesis tests
- Using help and syntax along the way

5.1.2 Starting Up

Starting Up

- Make a folder named `testscis`
- Start a log file called `testscis`
- Open up the ubiquitous auto dataset

```
. sysuse auto
```

5.2 CIs and Simple Hypothesis Tests

5.2.1 CIs

Confidence Intervals

- Try looking in help: `help ci`
  - `ci` follows standard syntax
  - We need to specify the kind of variable
- A CI for a continuous variable
- A CI for a 0/1 variable

```
. ci foreign, binomial

                                                    -- Binomial Exact --
    Variable |        Obs        Mean    Std. Err.       [95% Conf. Interval]
-------------+---------------------------------------------------------------
     foreign |         74    .2972973    .0531331         .196584    .4148353
```

- Easy enough from the Command window

Immediate CIs

- Stata will compute CIs without data
- This is an example of an *immediate* command
- The syntax is given in the help page, but the menus are simpler
  - Start here **Statistics > Summaries, tables and tests > Summary and descriptive statistics**
  - Pick one of the confidence intervals, say one for 2 successes in 50 trials

```
. cii 50 2

                                                    -- Binomial Exact --
    Variable |        Obs        Mean    Std. Err.       [95% Conf. Interval]
-------------+---------------------------------------------------------------
             |         50         .04    .0277128        .0048814    .1371376
```

5.2.2 Standard Hypothesis Tests

Differences in Means

- The menu nomenclature is accurate but non-standard, so first look at `help ttest`
- Stata needs to distinguish among
  - A single variable holding the measurement of interest, along with a variable holding the group identifier
  - Two variables holding measurements of interest
  - Two variables holding measurements of interest when observations do not correspond to individuals
    - This is the unpaired test, and should rarely bee needed

---

Example

- If we'd like to test for different mean gas mileages between foreign and domestic cars

```
. ttest mpg, by(foreign)

Two-sample t test with equal variances
------------------------------------------------------------------------------
   Group |     Obs        Mean    Std. Err.   Std. Dev.   [95% Conf. Interval]
---------+--------------------------------------------------------------------
Domestic |      52    19.82692     .657777    4.743297    18.50638    21.14747
 Foreign |      22    24.77273     1.40951    6.611187    21.84149    27.70396
---------+--------------------------------------------------------------------
combined |      74     21.2973    .6725511    5.785503     19.9569    22.63769
---------+--------------------------------------------------------------------
    diff |           -4.945804    1.362162               -7.661225   -2.230384
------------------------------------------------------------------------------
    diff = mean(Domestic) - mean(Foreign)                         t =  -3.6308
Ho: diff = 0                                     degrees of freedom =       72

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

- Output shows all three possible hypothesis tests
- It looks like the assumption of equal variances in the two groups might be violated

Differences in Standard Deviations

- Look at `help sdtest`—same general idea

```
. sdtest mpg, by(foreign)

Variance ratio test
------------------------------------------------------------------------------
   Group |     Obs        Mean    Std. Err.   Std. Dev.   [95% Conf. Interval]
---------+--------------------------------------------------------------------
Domestic |      52    19.82692     .657777    4.743297    18.50638    21.14747
 Foreign |      22    24.77273     1.40951    6.611187    21.84149    27.70396
---------+--------------------------------------------------------------------
combined |      74     21.2973    .6725511    5.785503     19.9569    22.63769
------------------------------------------------------------------------------
    ratio = sd(Domestic) / sd(Foreign)                            f =   0.5148
Ho: ratio = 1                                    degrees of freedom =   51, 21

    Ha: ratio < 1                Ha: ratio != 1                 Ha: ratio > 1
  Pr(F < f) = 0.0275         2*Pr(F < f) = 0.0549           Pr(F > f) = 0.9725
```

- Same general output
- Could now try unequal variances
  - Retrieve the preceding `ttest` command, and add the `unequal` option

```
. ttest mpg, by(foreign) unequal

Two-sample t test with unequal variances
------------------------------------------------------------------------------
   Group |     Obs        Mean    Std. Err.   Std. Dev.   [95% Conf. Interval]
---------+--------------------------------------------------------------------
Domestic |      52    19.82692     .657777    4.743297    18.50638    21.14747
 Foreign |      22    24.77273     1.40951    6.611187    21.84149    27.70396
---------+--------------------------------------------------------------------
combined |      74     21.2973    .6725511    5.785503     19.9569    22.63769
---------+--------------------------------------------------------------------
    diff |           -4.945804    1.555438               -8.120053   -1.771556
------------------------------------------------------------------------------
    diff = mean(Domestic) - mean(Foreign)                         t =  -3.1797
Ho: diff = 0                     Satterthwaite's degrees of freedom =  30.5463

    Ha: diff < 0                 Ha: diff != 0                 Ha: diff > 0
 Pr(T < t) = 0.0017         Pr(|T| > |t|) = 0.0034          Pr(T > t) = 0.9983
```

5.3 Conclusion

5.3.1 Conclusion

Finishing up

- Close the log file

Lesson 6

Basic Data Manipulation

6.1 Introduction

6.1.1 Goals

Goals

- Learn commands for making new variables
- Use all the standard clauses
- Learn commands for recoding variables
- See some common error messages
- Learn commands for getting rid of variables
- Learn how to check assertions
- Learn how to save and use datasets
- Learn how to enter data

Starting Up

- Make a folder for this lecture, say `makingdata`
- Start a log file with the same name

Getting a Dataset

- Go to **File > Example Datasets**...
- Click the `Example datasets installed with Stata` link,
- Click the `use` link for the census data.
  - Note that the command issued was `webuse census.dta`
  - Using any of these datasets is possible using `webuse`

What to do?

- `codebook`, of course.
- There are some things which scream out for new variables, such as forming proportions and rates.

6.2 Working with Data

6.2.1 Basic Data Work

Creating a new variable: `generate`

- The main command used for creating new variables is the `generate` command.
- There are others, but this is the main builder.
- Go look at the help file.

Creating a Proportion

- Let's create a variable which has the proportion of residents who live in urban areas.
- This is simple enough:

```
. gen prop_urban = popurban/pop
```

- A word on abbreviations: `generate` is a lot to type; abbreviating it as `gen` is readable.

Investigating the Proportions

- We can `summarize` the new variable, using the `detail` for more info.

```
. sum prop_urban, detail
```

```
                        prop_urban
-------------------------------------------------------------
      Percentiles      Smallest
 1%     .3377319       .3377319
 5%     .4643773       .3617681
10%     .4774035       .4643773       Obs                  50
25%     .5411116       .4732155       Sum of Wgt.          50

50%      .670646                      Mean           .6694913
                       Largest       Std. Dev.      .1440956
75%     .8030769       .8651392
90%     .8496912       .8699789       Variance       .0207635
95%     .8699789       .8903645       Skewness      -.2422186
99%     .9129498       .9129498       Kurtosis       2.285554
```

- Would it be OK to say that the overall proportion living in cities is 66.9%?
  - Nope!
- We need to use a weight here:

```
. sum prop_urban [freq=pop], detail
```

```
                        prop_urban
-------------------------------------------------------------
      Percentiles      Smallest
 1%     .3617681       .3377319
 5%     .4799327       .3617681
10%     .5411116       .4643773       Obs           225907472
25%     .6600978       .4732155       Sum of Wgt.   225907472

50%     .7333331                      Mean           .7366408
                       Largest       Std. Dev.      .1287142
75%     .8426136       .8651392
90%     .9129498       .8699789       Variance       .0165674
95%     .9129498       .8903645       Skewness      -.5696812
99%     .9129498       .9129498       Kurtosis       2.678608
```

Tables and Frequencies

- We could also look at urbanization by region
- Once again, it is important to use weighting

```
. tab region [freq=pop], sum(prop_urban)
```

```
    Census |              Summary of prop_urban
    region |        Mean   Std. Dev.       Freq.        Obs.
-----------+--------------------------------------------------
        NE |   .79180464   .10514547    49135283    49135283
  N Cntrl  |   .70533038    .0791576    58865670    58865670
     South |   .66604097   .13395661    74734029    74734029
      West |   .83876199   .10317816    43172490    43172490
-----------+--------------------------------------------------
     Total |   .73664079   .12871422   2.259e+08   225907472
```

Listing Information

- Sometimes it is worth listing the data to the Results window rather than opening the Data Browser.
- The command for this is `list`
- Lets look at the states which are under 40% urban

```
. list state prop_urban if prop_urban < float(0.40)

     +-------------------------+
     | state          prop_u~n |
     |-------------------------|
45.  | Vermont        .3377319 |
48.  | W. Virginia    .3617681 |
     +-------------------------+
```

Looking at Urbanized States

- We can sort the data, and look at the biggest proportions:

```
. sort prop_urban
. list state prop_urban in -5/-1

     +-------------------------+
     | state          prop_u~n |
     |-------------------------|
46.  | Nevada          .853158 |
47.  | Hawaii         .8651392 |
48.  | Rhode Island   .8699789 |
49.  | New Jersey     .8903645 |
50.  | California     .9129498 |
     +-------------------------+
```

- What would have happened if there were missing data?

Making an Indicator Variable

- Suppose we decide that if under 60% of the people live in urban areas, the state is rural.
- We can generate an indicator variable for this using Stata's logic.

```
. gen rural = prop_urban < float(0.60)
```

- What could have been dangerous about what we just did?

  Answer: we've ignored missing values. In this case, if a proportion were missing, the state would be classified as non-rural. Of course, there are not missing proportions in this dataset, so it makes no difference, here.

Changing Values

- Suppose we decided that the cutoff should be 50% instead.
- Try to generate the `rural` variable again by reusing most of the command.
- We get an error, because `generate` can only create variables, not change them.
- To change values, use the `replace` command.

```
. replace rural = prop_urban < float(0.50)

(7 real changes made)
```

6.2.2 Creating Categories

Creating Categories—Ideas

- Suppose we would like to categorize the states by urbanization
- We'd like to break into classes by percentage urban
  - 0–50, 50–60, 60–70, 70–80, 80 plus
    - The sloppiness with the endpoints is intentional, and will be made precise below
- Stata has a command that can do this nicely: `recode`
  - From the name, and the help, it appears that this is meant for recoding class values—but it works for creating categories also

---

Creating Categories—Implementation

- We need to create a `pct_urban` variable to avoid problems with decimal/binary roundoff

```
. gen pct_urban = 100 * prop_urban
```

- We then use `recode` for continuous variables, also:

```
. recode pct_urban ///
.   0/50=1 50/60=2 60/70=3 70/80=4 80/100=5 ///
.   , gen(class_urb)

(50 differences between pct_urban and class_urb)
```

- We used the `generate` option to create a new variable
  - Otherwise, Stata would have overwritten `pct_urban`

Creating Categories—Checking

- To be sure that we have the proper encoding, make a table of summary statistics

```
. tabstat pct_urban, by(class_urb) s(mean min max)

Summary for variables: pct_urban
     by categories of: class_urb (RECODE of pct_urban)

class_urb |      mean       min       max
----------+--------------------------------
        1 |  43.99384  33.77319  48.76692
        2 |  53.47107  50.86852  58.62541
        3 |  65.12933  60.03544  69.29301
        4 |  74.11783  70.63641  79.64625
        5 |  84.94676  80.30769  91.29498
----------+--------------------------------
    Total |  66.94913  33.77319  91.29498
------------------------------------------
```

Creating Categories—Endpoints

- Stata generates the categories from left to right
- In the above example, this means that the *right* endpoints are included in each classification
- If we wanted the `left` endpoints included, we would right the list in the opposite order:

```
. recode pct_urban ///
.   80/100=5 70/80=4 60/70=3 50/60=2 0/50=1 ///
.   , gen(class_urb_2)
```

  - In this particular example it doesn't make a difference—but it could in others

---

Getting Rid of Variables

- To get rid of a particular variable, `drop` it.
- Here, we could get rid of our `class_urb_2` variable

```
. drop class_urb_2
```

  - This affects only the dataset in memory—the dataset stored on the disk is still in its original form

---

6.2.3 Confirming Assumptions

Confirming Assumptions with `assert`

- We would like to check the numbers: Do all age populations add up to the `pop`?
- The command for checking such guesses is `assert`
- Here, we can try

```
. assert pop==poplt5 + pop5_17 + pop18p + pop65p
```

!/swv-UOLBY38/out

- Oops... this assertion is very false.
  - Of course! 65 and over is a subset of 18 and over!
- Try number 2:

```
. assert pop==poplt5 + pop5_17 + pop18p
```

Being Careful with `assert`

- Remember Stata's logic: the assertion will be true for an observation if it evaluates to anything other than 0.
- Try the following assertion

```
. assert divorce
```

- Since all states have at least one divorce(!), `divorce` is always "true".

6.2.4 Saving and Using

Stata Being Careful

- We'd like to use another dataset for the next session
- Try loading some year 2000 census data:
- We get an error!
  - Stata will not load a new dataset over one whose data have changed
    - **Warning**: Stata's definition of data is strict: if you change metadata, namely value labels or the like, you still can bring in new data!

Saving Our Creation

- From Menus
  - **File > Save** ... will ask about overwriting the already-existing file
  - **File > Save As**... will ask for a new file name
- From the Command window
  - The `save` command is used. To overwrite a file, the `replace` option must be used.
- We have the proper working directory, so a relative path is good:

```
. save "altered census"

file altered census.dta saved
```

- To save it again:

```
. save "altered census", replace

file altered census.dta saved
```

Using our Creation

- We can clear the data from memory with the `clear` command. Do this now.
- We can now open our dataset by either using the **File > Open**... menu item, or the `use` command.

```
. use "altered census"
```

6.2.5 Subgroups and Extended Generation

More Ways to Create Variables: `egen`

- Stata has an extended generate command called `egen`
- It is made for creating variables within groups or across variables
- The `egen` command uses its own functions (called `fcns`) to generate the new variables
  - These pseudo-functions work only with `egen`
- Take a look a the help

Working Across Variables

- Bring in the `pop2000` dataset which comes with Stata

```
. sysuse pop2000
```

- We can quickly sum up the total male population

```
. egen male_ttl = rowtotal(malewhite-maleisland)
```

- Suppose we would like to check the totals here, too

```
. assert male_ttl == maletotal
```

  - As it turns out, `male_ttl` doesn't ever match `maletotal`

Notes about the `row... fcns`

- These are made for summing up questionnaire responses
- Because of this, missing values do *not* lead to missing totals, averages, etc.
  - When computing averages, the computed average is the average across all non-missing values

---

Working Within Groups—Ideas

- Many of `egen`'s functions are made to generate values within groups
- Notice in the help that it talks about using `by`
- We ran into this when making tables—this refers to using the `by` prefix command

---

Working Within Groups—Example

- We would like to use the `auto` dataset, but our current dataset is dirty
- So... clear first, then grab the `auto` dataset

```
. clear
. sysuse auto
```

- Suppose we would like to store the mean value of mpg within each of the repair categories *as a new variable*
- We'll use `by rep78` to define the groups, then use egen

```
. by rep78, sort: egen mpg_mean = mean(mpg)
```

Notes about `egen`

- Many `egen` of the `egen` commands allow the `by` prefix; some do not
- Remember that the `by` prefix can be used with a variable list
  - This creates a subgroup for each possible set of values for the variables

---

Notes about Data Manipulation

- We've covered some of the main tools for creating new from old
- There are more things we still need to cover
  - Converting categorical string variables to numerical variables
  - Converting string variables containing dates or formatted numbers into numerical variables Stata can use
- These will be covered when talking about cleaning up a dataset

---

6.3 Stata's Data Editor

6.3.1 Entering Data Into Stata

Entering Data into Stata

- The editor in Stata can be used to
  - Enter new data.
  - Edit existing data.
  - Rename Variables.
  - Relabel Variables.
  - Change display formats.
  - The latter three are done by double-clicking anywhere in the column, and filling in the information in the dialog box.

Using the Editor to Enter Data

- Click on the Editor button.
  - The `edit` command has the same effect.
- Enter data
  - Tab to the 'next cell'
    - When entering the first observation of a new dataset, the next cell is the next cell to the right.
    - When entering or editing data in an existing dataset, the tab key is smart enough to wrap at the last variable.
  - Enter key to move down.
  - Missing data can simply be skipped (no entry required).

What about Undo?

- The editor has an uncommon method for undo:
  - To commit (i.e. make permanent) all changes, click the Preserve button.
    - This cannot be undone!
  - To undo all changes since the last preserve, click the Restore button.
    - This cannot be redone!
- This is a rather draconian treatment of undoing actions!

Judgement: Data Entry in Stata

- The editor is useful for renaming & relabeling.
- The editor is useful for small datasets.
- It is better to use something else for data entry of any other datasets.
  - Spreadsheets
  - Databases (much better)

6.4 Conclusion

6.4.1 Conclusion

Finishing Up

- Close your log file
- Clear out the dataset

Final Thoughts

- We've seen some of the many tools for data creation
- There are many, many others....

Lesson 7

Getting Tabular Data into Stata

7.1 Introduction

7.1.1 Goals

Goals

- Import data from a spreadsheet
- Learn how importing works from other packages

7.1.2 Getting Started

Usual Startup

- We'll be using data from a different location in this lesson
- Make a folder for the lesson: `datafromtables`
- Start a log file in the folder, say `datafromtables`
- We'll want a clean slate to work with, so clear out the data in memory

```
. clear
```

Unusual Startup

- We need to go get the files for this lecture
- Type this into the Command window:
  `net from http://www.stata.com/training/09apr2009`
- Click the link `datafromtables`
- Click the link `click here to get` to download the data to your current working directory

---

7.2 Importing Tabular Data

7.2.1 Importing Tabular Data—Copy and Paste

What are Tabular Data?

- A single worksheet in an MS Excel Workbook.
- A table exported from a database.
- Any text file where there are clear column delimiters, and there is one observation per row.

Moving Data from a Spreadsheet to Stata using the Editor

- Data can be brought in from any spreadsheet by
  - copying from the spreadsheet, and
  - pasting into Stata's editor
- Some guidelines to follow:
  - Have a single row containing nice column headers
    - These will be used for the variable labels and for the variable names (as best possible)
  - It helps to have a numerical variable as the first column

---

The Data to Import

- Open the file `sabbatical_apt.xls` from within MS Excel (or OpenOffice)

	A	B	C	D	E	F
1	Location	Quality	Exterior	Rent	Available	Number of Rooms
2	3	poor	brick	$720	1/18/2008	1
3	2	poor	siding	$1,400	1/1/2008	2
4	1	perfect	clapboard	$985	1/27/2008	3
5	1	good	brick	$1,500	2/17/2008	4
6	0	perfect	shingle	$2,100	1/25/2008	4
7	1	good	brick	$1,100	12/27/2007	2
8	4	poor	clapboard	$800	1/20/2008	2

(The locations for the apartments are on the second tab.)

Example - Steps

- Steps for bringing the data into Stata.
  - Copy all the data.
  - Switch to Stata.
    - If you neglected to clear the data earlier, do so now.
  - Open the Editor.
  - Paste the data.

What Happens?

- Get a quick look at the structure to see that
  - The header row is used to form the variable descriptions.
  - The header row is used to form variable names.
    - All illegal characters are removed.
    - All variables names are lowercase.
    - The first 32 characters are used.
    - If 32 characters aren't enough to make a unique name, it will make a generic name.
  - Dollar signs and commas interfere with numbers
  - Dates are brought in as strings
    - These will be fixed later.

Judgement

- This method is very simple and useful, since it
  - allows spreadsheets to be used to gather data.
  - allows spreadsheets to be used to share data.
- In this example, MS Excel was used as the spreadsheet application.
  - Anything which keeps its data as a table, and allows copying will work.
- This limits reproducibility, though.

What if Copying and Pasting are not Possible?

- Cutting and pasting of data may not be possible, depending on the data source.
  - For example, when taking data from one statistical package to another.
- Cutting and pasting can be problematic if there are too many data.
- Need a method to get the data out of one application and into another application.
  - This is known as exporting and then importing the data.

7.2.2 Importing Tabular Data Using the insheet Command

Exporting & Importing

- Every application has its own data format.
  - For example, Stata cannot directly read Excel data, and vice versa.
- To bring in data which were created in another application, the data need to be imported.
  - This can be done only if the importing application has a data translation tool built in!
- If the data cannot be directly imported, the data first must be exported by the creating application into a common format, after which it can be imported by the application wishing to use the data.

What is the Common Format?

- The two most common formats for exchanging data are
  - Tab delimited files, where there are Tab characters between items of information.
  - Comma delimited files, where there are commas between items of information.
  - Note: In both these formats, there should be double quotes around strings, but not all applications abide by this rule.

Our Example for Exporting and Importing

- We'll use MS Excel to illustrate this, but it applies to a wider class of applications...
- Steps which need to be done:
  - Move the data out of the place it is stored and into a common format.
  - Bring the data from the common format into Stata.

Exporting from a Spreadsheet

- Change the format of all numbers (other than dates) to General.
  - This is not strictly necessary, since we'll see a way to reformat numbers containing odd characters properly later. It can help for long decimals, though.
- Go to **File > Save As**... and choose **Text (Tab delimited)**
  - Only one sheet may be exported at a time.

When Does It Work Well?

- The export will work best when
  - the variable names are in the first row
    - otherwise the Stata variables will have meaningless names.
  - blank lines are removed first
    - otherwise there will be observations made of missing data
  - there is only one table per sheet
    - otherwise the data from the tables will be mixed together.

Importing into Stata—the `insheet` command

- Bring the data into Stata using the `insheet` command
  - Look at `help insheet`
  - `insheet using "`*filename*`" [, options]`
  - There are several options depending on the dataset.
  - The `clear` option will clear out memory.
    - It is safer to consciously do this on a separate line.

For Our Example

- If the current working directory is set up properly, we can type a simple command:

```
insheet using "sabbatical_apt.txt"
```

`insheet` Example - Steps

- Switch to Excel.
  - The `sabbatical_apt.xls` file should still be there.
- Save the file as a text (tab delimited) file.
- Switch to Stata.
  - Be sure that there are no data in use.
- Type `insheet using "`*filename*`"` into the Command window.

`insheet` Example - Notes

- This looks very similar to the result of cutting and pasting (as it should).

Judgement

- This method is a useful general method, but requires care.
  - It is easier than cutting and pasting for large amounts of data.
  - It is more general than cutting and pasting, because many different types of software understand how to make either comma-delimited or tab-delimited files.
  - It is completely reproducible.
  - It is clumsier than cutting and pasting for small spreadsheets.
  - If a worksheet contains many tables (as it shouldn't), there will be a mess.
- Remember that the `insheet` command can be used with any tab- or comma-delimited file!

  To be more accurate: it can be used with any file that has a consistent delimiter, because of the `delimiter` option. Most data files nowadays, however, have either tabs or commas for delimiters.

7.2.3 Working with Other Stats Packages or Databases

SAS

- Stata and SAS can trade data using SAS `xport` files.
- `fdause` can read `xport` files, `fdasave` can save `xport` files.

SPSS

- SPSS can write Stata 9 files.
  - From within SPSS use **File > Save As...** to save the file as a Stata file
  - `use` the dataset in Stata.
- SPSS can read Stata 9 files
  - From within Stata, `save` *filename*`, old` will save the file in Stata 9 format.
  - From within SPSS use **File > Open > Data...**, pick the Stata file type, and open the file.

Using a Translation Utility

- StatTransfer is a utility which can translate many different data formats.
- This can be worthwhile if you need to translate data often.

Using ODBC

- Stata can use ODBC (Open Database Connectivity) to extract or write data from a database
- Look at `help odbc`
- We will not cover the details here

7.3 Conclusion

7.3.1 Conclusion

Finishing Up

- Close your log file
- Clear out the memory

Conclusion

- Transferring data from one statistical package to another is worthwhile.
- Using export/import is better with spreadsheets.
- We need tools for working with formatted numbers and dates

Lesson 8

From Data to Dataset

8.1 Introduction

8.1.1 Goals

Goals

- Making a step towards automation
- Learning how to document a dataset
- Learning about labels
- Learning about encoding
- Learning about dates and times

Startup

- Make a new folder called `cleaningup`
- Start a log
- Grab the `cleaningup` package from the Stata web site

8.2 A Step Towards Automation

8.2.1 Working with Command logs

Command Logs?

no; data in memory lost
clear
use "sabbatical-apt"

- A command log contains the commands corresponding to the results window, but not output.
- These are useful for later replication of a session.
  - There are no leading dot prompts, which is a help.
  - They can be used for automation quite easilty.
- We'll be doing some complicated work, here, so we would like to keep the commands.

The `cmdlog` Command

programmer's tool

- Look at `help cmdlog`
  - Very similar to the log command, though there is no menu item.
- Start a command log named `cleaningup`.
- We'll turn this into a do-file at the end of the lesson.

Thinking Ahead and Starting

- We'll want to pare out bad commands from the command log, later.
- So... put in a comment after bad or worthless commands, or before commands which you know you will want to ignore
- Open up the `sabbatical_apt` dataset we'll be cleaning up:

```
. use "sabbatical_apt"
```

8.3 From Data to Dataset

8.3.1 General Dataset Goals

From Data to Dataset

- Once the data are in Stata, there is still a need to make the dataset useable in case
  - it is passed on to another researcher, or
  - you put it away for several months.
- The goal is to get the dataset
  - self contained, so that no external code books, notes, or explanations will be needed, and
  - ready for analysis
- Documentation is key!

Order of Preparation, Part I

- Make a copy of the original dataset!
  This can be done through Windows, or this can be done in Command window with the odd command
  `copy "original file name" "copy's file name"`
- Document the dataset itself.
  - Add any necessary descriptions & notes.
- Go through the dataset variable by variable.
  - Recode variables which are particularly badly coded.
  - Store variables needed for analysis as numbers:
    - Encode categorical variables, making sure that encoded variables are comprehensible without a book.
    - Convert dates or times to elapsed dates for computation.
  - Add descriptions and notes.

Order of Preparation, Part II

- Spruce up the dataset.
  - Drop any old, unnecessary variables.
  - Be sure that the remaining variables are properly named and ordered
  - Much of this can be done by right clicking on variables in the Variables window
    - Nice for small datasets, awful for large datasets
- Store the data efficiently.

Saving the Dataset

- Save the dataset after each stage, using a new name.
- This helps when looking at what was done.
- The temporary files can later be deleted.

A Note on Validation

- We will start with the assumption that the data are reasonably good.
- This is an awful assumption—we're doing it to build up concepts which will help for reading in data.
- We'll get back briefly to validation when we get to automation.

Let's Get Started

- Run a codebook command to be sure we know what to do

```
. codebook
```

```
-------------------------------------------------------------------------------
location                                                             Location
-------------------------------------------------------------------------------

                  type:  numeric (byte)

                 range:  [0,4]                        units:  1
         unique values:  5                        missing .:  0/7

            tabulation:  Freq.  Value
                            1  0
                            3  1
                            1  2
                            1  3
                            1  4

-------------------------------------------------------------------------------
quality                                                               Quality
-------------------------------------------------------------------------------

                  type:  string (str7)

         unique values:  3                        missing "":  0/7

            tabulation:  Freq.  Value
                            2  "good"
                            2  "perfect"
                            3  "poor"

-------------------------------------------------------------------------------
exterior                                                             Exterior
-------------------------------------------------------------------------------

                  type:  string (str9)

         unique values:  4                        missing "":  0/7

            tabulation:  Freq.  Value
                            3  "brick"
                            2  "clapboard"
                            1  "shingle"
                            1  "siding"

-------------------------------------------------------------------------------
rent                                                                     Rent
-------------------------------------------------------------------------------

                  type:  string (str7)

         unique values:  7                        missing "":  0/7
```

```
              tabulation:  Freq.  Value
                               1  "$1,100 "
                               1  "$1,400 "
                               1  "$1,500 "
                               1  "$2,100 "
                               1  "$720 "
                               1  "$800 "
                               1  "$985 "

                 warning:  variable has trailing blanks

-------------------------------------------------------------------------------
available                                                           Available
-------------------------------------------------------------------------------

                    type:  string (str10)

          unique values:  7                        missing "":  0/7

              tabulation:  Freq.  Value
                               1  "1/1/2008"
                               1  "1/18/2008"
                               1  "1/20/2008"
                               1  "1/25/2008"
                               1  "1/27/2008"
                               1  "12/27/2007"
                               1  "2/17/2008"

-------------------------------------------------------------------------------
numberofrooms                                                 Number of Rooms
-------------------------------------------------------------------------------

                    type:  numeric (byte)

                   range:  [1,4]                         units:  1
          unique values:  4                          missing .:  0/7

              tabulation:  Freq.  Value
                               1  1
                               3  2
                               1  3
                               2  4
```

8.3.2 Notes

Adding Notes

- Notes can be added by right-clicking a variable name
- The `notes` command is useful for adding notes about the dataset and/or the variables:
  `notes [varname]: put your note here`

  - Without a variable, the note pertains to the dataset.
  - With a variable, the note pertains to the variable.
  - To time-stamp the note, include the word TS surrounded by blanks.
- Attach some notes to the dataset now!

Viewing and Deleting Notes

- To see all the notes about everything, type

  `notes`
- Now look at the help for `notes`
  - The *evarlist* is a *varlist* with the added "variable" `_dta` for notes which pertain to the dataset itself.

(only given the codebook command)

Notes about Notes

- Good notes to add to the dataset and/or variables
  - The origin of the data.
  - The contact for the dataset maintenance.
  - Any notes about data collection quirks.
- Notes attached to a variable are attached to the variable itself, thus making the variable self-documenting if it is ever brought into another dataset!
  - Tremendously useful!

8.3.3 Labels

Give the Dataset a Description

- In Stata parlance, descriptions are called labels.
  - So... we are looking for a data label.
- Here's the command:
  - `label data "`*put in a data label*`"`
  - Check the help for all of the types of label commands!
- Shows up in the results from describe.
- Comes up on screen when the data are brought into Stata.
- Was more useful when file names were shorter.

Giving Variables Descriptions

- Most datasets require some variables be replaced or recoded.
- Thus, the variable descriptions should be made at the end of the cleanup.
  - Unless you would like them as reminders.

Categorical Variables

- Categorical Variables have values (categories) which are not intrinsically numerical.
  - The categories can be unordered, such as "plumber", "nurse", and "taxi driver".
  - The categories can be ordered, such as "small", "medium", "large".
  - Anti-example: identification "numbers" like "KH432"
    - These are not categories!

Storing Categories as Numbers

- It is almost always easier to manipulate variables in a statistics package when they are stored as numbers.
  - Regrouping and changing order in listings is much simpler
- Categorical variables which are stored as numbers are called encoded variables.
  - To make them friendly, they should be readable to humans.
  - To make these readable, Stata uses value labels.

In the Sabbatical Apt Search, What Type of Categorical Variables Are There?

- The location variable is already encoded, because it is stored as a number.
  - It is not yet readable, though.
- The exterior variable is categorical, unencoded, and unordered.
- The quality variable is categorical, unencoded, and ordered.
- Strangely enough, these are the three different cases which can arise!

What Are Value Labels?

- Value labels allow encoded variables to be readable.
  - In keeping with Stata terminology, think of the value labels as descriptions of the values of an encoded variable.
- How do value labels work?
  - A value label has a name of at most 32 characters.
  - A value label consists of a list of pairs of numbers and tags called value-tag pairs.
  - The values must be integers, though negative numbers are OK
  - There can be 64K different values
  - Each value is associated with a tag of up to 32,000 characters.

Case 1: Making an Encoded Variable Readable

- Making an encoded variable readable needs two steps:
  - Defining the value label
  - Attaching the value label to the variable.
- Note: one value label may be attached to many variables, so the first step above may not need to be done for every variable.
  - This is useful when many variables are encoded in the same way, such as ranking questions in a questionnaire.

Defining a Value Label

- Defining a value label requires that each value be associated with a tag:

  `label define` *lblname* `#` `"`*tag*`"` `[# "`*tag*`" ...][, modify]`

General Examples:

- `label define noyes 0 "No" 1 "Yes"`
  - Note: 'noyes' is a good name for a label, since it clearly states the label's meaning.
- `lab def nlbl 0 "No" 1 "Yes"`
  - Note: 'nlbl' is a bad name for a label, since it is incomprehensible as a name.
- `label def noyes 2 "Maybe", modify`
  - This modifies the noyes label by adding a value-tag pair

Attaching a Value Label to a Variable

- This is done by:
  `label values` *varname lblname*
- General Examples:
  - `label values religion rlbl`
  - `label values married noyes`
  - `label val diabetic noyes`
- Note: one value label may be attached to many variables!

In the Sabbatical Apt Search: Making `location` Readable

- The `location` variable is already encoded, but not yet readable.
- Make a value label, purposely misspelling one tag :

```
. label define whereat ///
. 0 "Central" 1 "North" 2 "South" 3 "East" 4 "Weste"
```

- Attach the `whereat` label to the `location` variable:

```
. label val location whereat
```

Checking Results and Fixing

- Check our result

```
. codebook location
```

```
-------------------------------------------------------------------------------
location                                                              Location
-------------------------------------------------------------------------------

                  type:  numeric (byte)
                 label:  whereat

                 range:  [0,4]                        units:  1
         unique values:  5                        missing .:  0/7

            tabulation:  Freq.   Numeric  Label
                            1         0  Central
                            3         1  North
                            1         2  South
                            1         3  East
                            1         4  Weste
```

- Fix the tag:

```
. label def whereat 4 "West", modify
```

Notes on Naming Value Labels

- In this example the variable (location) and the value label (whereat) have different names. They could have had the same names without any confusion.
  - Variables and value labels are different objects in the dataset, hence Stata will not mistake one for the other.

Small Notes about Value Labels

- To get rid of an unwanted value label, use
- When exiting, Stata throws away all unattached value labels by default.
  - So... if you've defined some value labels, either attach them before quitting, or save the dataset with the `orphans` option.
- To unattach a value label, use

  `label values` *varname*
  - The label will still exist unless the dataset is saved.
- It is possible to label missing values, since the missing values are (technically) integers.

Case 2: Encoding Unordered Categories

- Stata can encode string variables quite easily, if there is no natural value order:

  `encode` *varname*`, generate(`*newvar*`)`
- The values will be put in alphabetical order, and then encoded with integers starting at 1.

  This is not quite true, because they are put in ASCII order, so that capital Z comes before small a. With similar captialization, though, they are in alphabetical order.
- A new value label will be defined whose name is that of the *newvar*.
- The new value label will be attached to the newly created variable.

In the Sabbatical Apt Search: Encoding the exterior Variable

- This is simple

```
. encode exterior, generate(exterior_enc)
```

- Look at the result

```
. codebook exterior_enc
```

```
-------------------------------------------------------------------------------
exterior_enc                                                         Exterior
-------------------------------------------------------------------------------

                  type:  numeric (long)
                 label:  exterior_enc

                 range:  [1,4]                        units:  1
         unique values:  4                        missing .:  0/7

            tabulation:  Freq.   Numeric  Label
                            3         1  brick
                            2         2  clapboard
                            1         3  shingle
                            1         4  siding
```

- Notice that there is just one step in this process!

What Order Does Stata Use to Create the Pairs?

- The default ordering here is just fine, because the categories are unordered.
- This is dangerous when using datasets which arrive sequentially, because the value-tag pairs can change if a value is missing or changed.
  - Imagine what would happen if the `clapboard` value were not represented...

Putting New Variables Where They Belong

- After creating a new variable whose purpose is to take the place of an old variable,
  - Be sure that the changes are correct.
  - Use the `move` command to move the new variable in front of the old variable
    `move` *newvar oldvar*
- For the example here:
  - Tab completion is a real friend here
- This ensures that the variables in the final dataset will be in the proper order after cleaning up.

Case 3: Encoding Ordered Categories

- To encode a string variable which has a natural order, two steps are required:
  - A value label containing the properly ordered value-tag pairs must be defined.
    - Note that case is important! "Yes" is different from "yes"!
  - The `encode` command must then be used with the `label` option, so that it will use this value label for the encoding

    `encode` *varname*`, generate(`*newvar*`) label(`*lblname*`)`

In the Sabbatical Apt Search, Encoding the `quality` Variable

- The quality variable contains string values of "poor", "good", and "perfect".
- To encode the variable
  - Create a good label

    ```
    . label def qual 0 "poor" 1 "good" 2 "perfect"
    ```
  - Use it to create a new variable

    ```
    . encode quality, gen(quality_enc) label(qual)
    ```
  - Choosing 0 for the first category can help in listing acceptable apartments
- Finish up by moving the `quality_enc` variable in front of the quality variable.

  ```
  . move quality_enc quality
  ```

Notes on Ordered Encoding

- This is safer than the unordered encoding, because the categories are fixed
- Some care is needed, because Stata cares about capitalization
  - Using `Poor` instead of `poor` would cause trouble
- Any integers could be used for the value label

8.3.4 Working with Dates

Working with Dates

- When importing dates into Stata, the dates are initially stored as strings
- Converting a string date to an numerical date takes two steps
  - The date must be converted to a number using the `date()` function
  - The numerical value must be made human-readable using formatting
- We should go look at help for dates
  - This is a bit over the top with explanations—click on the blue `here` in `Click here to skip...`
    - We still end up with very technical wording

The `date()` Function

- The `date()` function looks like:

  `date(`*string*`,` *date mask* `[,` *century* `])`

- The *string* in the first argument is typically a string variable
- The *date mask* tells Stata the order of the day, month, and year
- The *century* is for two-year dates

Four Year Date Examples:

- Suppose we had a string variable `strdate` with string dates in it
- If the dates looked like "January 20, 2009", we would use

  ```
  gen numdate = date(strdate, "MDY")
  ```

- If the dates looked like "20.jan.2009", we would use

  ```
  gen numdate = date(strdate,"DMY")
  ```

- If we had metric dates, such as "20090120" we would use

  ```
  gen numdate = date(strdate,"YMD")
  ```

Capitalization intentional

Two Year Dates: Rules

- With two-year dates, there are two choices:
  - A two-digit century can be placed before the token for the year if all the two-year dates come from the same century.
  - A third argument can be given for the maximum year.

Two Year Dates: Examples

- If the dates lookedlike "14july89", and all dates were from the 1900's, we would use

```
gen numdate = date(strdate, "DM19Y")
```

- If the same short dates contained dates from two centuries, but no dates were later than 2009, we would use

```
gen numdate = date(strdate, "DMY", 2009)
```

  - So a year of 09 would be considered 2009, while a year of 10 would be considered 1910.

Converting `available`

- Here, the date variable is month/day/year, with proper 4-digit years, so the command is

```
. gen available_num = date(available, "MDY")
```

- Look in the browser, and note that the date is not readable!

```
. browse availble_num
```

- We can change the format to Stata's default easily:

```
. format %td available_num
```

Looking at the Dates

- We can check the look by listing the variable

```
. list available*
```

```
     +-----------------------+
     |  available   availab~m |
     |-----------------------|
  1. |  1/18/2008   18jan2008 |
  2. |   1/1/2008   01jan2008 |
  3. |  1/27/2008   27jan2008 |
  4. |  2/17/2008   17feb2008 |
  5. |  1/25/2008   25jan2008 |
     |-----------------------|
  6. | 12/27/2007   27dec2007 |
  7. |  1/20/2008   20jan2008 |
     +-----------------------+
```

- If we would like another format, there are plenty
  - Look at `help dates`, and click the link for formatting
- If we would like more standard dates

```
. format %tdMon_dd,_CCYY available_num
. list available*
```

```
     +---------------------------+
     |  available   available_~m |
     |---------------------------|
  1. |  1/18/2008   Jan 18, 2008 |
  2. |   1/1/2008    Jan 1, 2008 |
  3. |  1/27/2008   Jan 27, 2008 |
  4. |  2/17/2008   Feb 17, 2008 |
  5. |  1/25/2008   Jan 25, 2008 |
     |---------------------------|
  6. | 12/27/2007   Dec 27, 2007 |
  7. |  1/20/2008   Jan 20, 2008 |
     +---------------------------+
```

Finishing Up with Dates

- We need to move the variable to its new location

```
. move available_num available
```

- We could give the new variable a nice label here

```
. label var available_num "Date available"
```

  - We could also wait, and put the variable label on at the last step—when this gets done depends on whether the variable will need to be manipulated again or not

---

8.3.5 Fixing Nasty Variables

Look at the `rent` Variable

- It is useless as a number, because of the dollar signs and commas.
- There is a nice command made just for this: `destring`
- Take a look at its help.

Using `destring` for these Data

- The problem characters in the `rent` variable are `$` and `,`.
- We should make a new variable to be sure that all things match well.
- Here is the command

```
. destring rent, gen(rent_num) ignore("\$,")
rent: characters $ , removed; rent_num generated as int
```

- If the resulting variable still has some values which are not numeric, Stata issues a warning and does not generate the new variable.

```
. destring rent, gen(ohno) ignore("$")
```

  - This behavior can be overridden with the `force` options.
- No need to move this variable!

Notes on Nastiness

- The `tostring` command can be useful for forcing numerical id numbers to be strings (as they should be).
- The `filefilter` command is useful when there are troubles caused by odd file formats, such as multicharacter delimiters or odd non-displayable characters.
- When all else fails, but you think you can work through the original file line by line, `file` is an advanced, but very flexible command.

Missing Values

- Encoded missing values can be changed to true missing values using `mvdecode`
  - If, say, 98 were "did not answer", and 99 were "not applicable", these could be recoded using `mvdecode` *varname* `, mv(98=.a 99=.b)`

8.3.6 Cleaning Up

Keeping Just What is Needed—Idea

- We're now ready to throw away the old variables which we've cleaned up
- `keep` and `drop` are used to pare out observations or variables in a dataset.
- When cleaning datasets `drop` is safer, because it forces a conscious decision about which variables should be dumped.
- `keep` is best used when explicitly retaining a particular subset for an analysis.

Keeping Just What is Needed—Example

- Think Ahead! The following steps will destroy data, so save a copy of the data set first under a new name!

```
. save "sabbatical_apt_predrop"

file sabbatical_apt_predrop.dta saved
```

- Now throw away the unneeded variables

```
. drop exterior quality available rent
```

Changing Variable Names

- `rename` *varname newvarname* renames variables.
- For us:

```
. rename exterior_enc exterior
. rename quality_enc quality
. rename available_num available_date
. rename rent_num rent
. rename numberofrooms rooms
```

- This can be done via right-clicks if you like

A Note on Batch Renaming

- Stata has a utility called `renpfix` for replacing common prefixes in variable names
- There is a more flexible utility called `renvars`
  - Try `findit renvars`
- These are useful for large, wide datasets

Changing Variable Labels

- To change a variable description, use the `label variable` command:
  `label variable` *varname* `"`*description of the variable here*`"`
- Here, most of the variables kept good variable labels
- We've got nothing to do...

Finishing Touches

- If not done already, the data can be reordered.
- `order` *varlist*
  - makes the dataset begin with the variables in the varlist.
- Store the data more efficiently by using the `compress` command

```
. compress

quality was long now byte
available_date was float now int
exterior was long now byte
```

- This simply saves the data as efficiently as possible

Saving the Dataset

- Now we are ready to save the fruits of our labor

```
. save "sabbatical_apt_cleaned"

file sabbatical_apt_cleaned.dta saved
```

- Close your command log here, so it contains only commands you need to clean up the dataset.

```
. cmdlog close
```

What a Load of Work!

- Don't dismay, this work is worth the effort.
  - The data can now be shared.
  - The most likely person with whom you will share the data is your future self.
- If someone else takes care of your data, expect this type of meticulous care.

8.4 Command Log to Do-File

8.4.1 Command Log to Do-File

Opening the Command Log

- Start the Do-file Editor by clicking steno pad icon
  - Or type `doedit` in the Command window
- Use the menubar in the Do-file Editor to go to **File > Open...**
- Be sure you view files of type Text
- Open up `cleaningup.txt`

What Have We Here?

- The command log is simply a series of Stata commands
- If we keep just the worthwhile command, we'll have a file which can reproduce cleaning the dataset
  - We should remove all errors
  - We should add useful comments

Running the Do-file

- Save the file as `cleaningup.do`
- Try "doing" the file by clicking the **Do** button (shaded paper)
  - Clicking the **Run** button will run the commands silently
- This should recreate the cleaned-up file
- From the Command window: `do cleaningup`
  - Stata adds the `.do` extension
  - Stata then looks in the working directory to find the do-file

---

What About Errors?

- If you get errors when doing your file, you'll need to find the command creating the error and fix it
- It is also possible to force the do-file to run by giving the `nostop` option:

```
do cleaningup, nostop
```

- Note: Using the `nostop` option can be risky if datasets are changed

---

Workflow Note

- Using a command log to create a do-file is useful for beginners
- As you get used to using the Command window, it is more typical to write the do-file directly
- We will see a way to write more complete do-files in a later lesson
  - We'd like them to keep their own logs, for example
  - We need them to be robust against programming errors, too

---

8.5 Conclusion

8.5.1 Conclusion

Additional Notes about Cleaning Up

- There are some utilities we did not see in this variable

Finishing up

- Be sure to close your log file

8.5.2 What was done?

Much good work

- We cleaned up a dataset
- We learned to work with dates

  Stata 10 has many routines for working with date-time timestamps, even going so far as to use the two different ways of tracking time nowadays—with and without leap seconds.
- We learned how to fix up ugly formatting of nice numbers.
- We learned how to use a command log to create a do-file which could then be used to reproduce our work

---

Lesson 9

Combining Datasets

9.1 Introduction

9.1.1 Goals

Goals

- Combining Datasets.
  - Adding new data to old - adding observations.
    - Arises when data arrive periodically and are accumulated into one dataset.
  - Combining data from many sources - adding variables.

9.1.2 Getting Started

Getting Started

- Make a new folder called `combiningdata`
- Start a log file, also called `combiningdata`
- Grab the data from the `combining` package for the training site
  - Typing `net` might work...

9.2 Combining Datasets

9.2.1 Background

Example

- Have some clinic information:
  - Old visit information in one file.
  - New visit information in another file.
  - Information about the patients in a third file
- Want:
  - Put all the visit information together
  - Bring in the information about the patients

The Data

	id	visit	illness	insurance
1	9	26dec2006	Cold	Major Med
2	4	09jan2007	Sore Throat	HMO
3	1	10jan2007	Pneu	.
4	25	02feb2007	Cold	PPO
5	4	05feb2007	Sore Throat	.
6	25	21apr2007	Cold	PPO
7	9	20may2007	Flu	.

Old Visit Info

New Visit Info

	id	visit	illness	insurance	doctor
1	616	24apr2007	Pneu	HMO	Jones
2	9	30may2007	Sore Throat	HMO	Smith

Patient Info

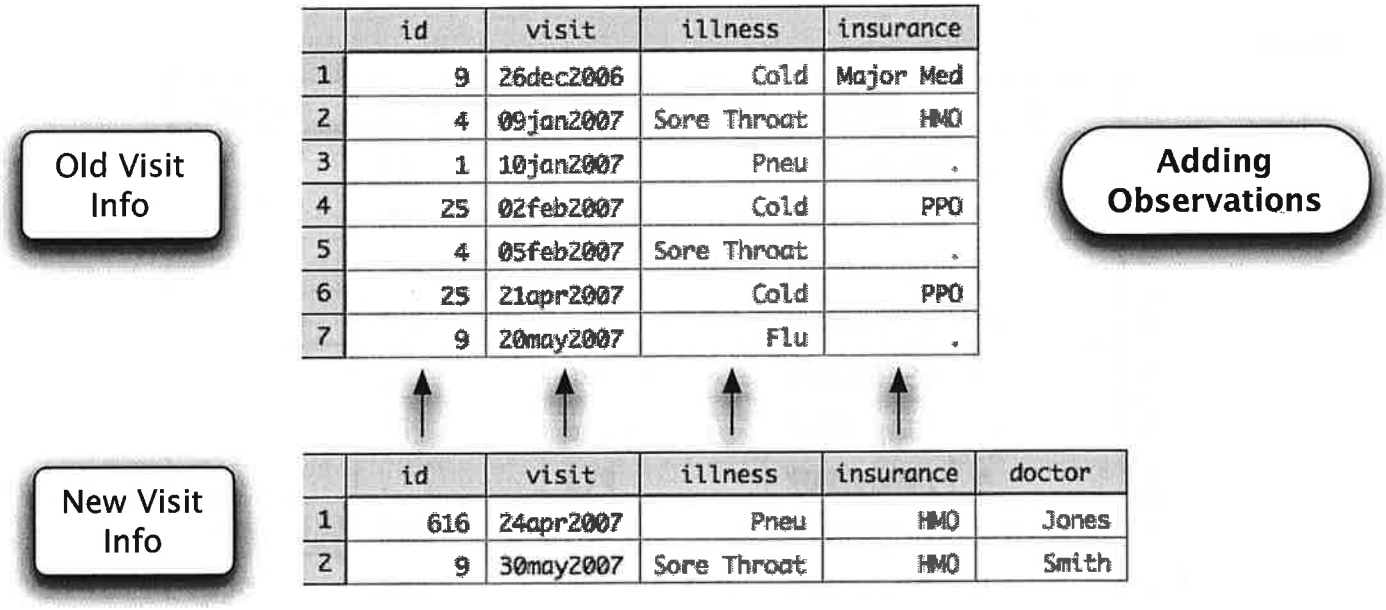
	id	birthdate	gender	insurance
1	1	13apr1995	Female	PPO
2	4	21oct1989	Male	HMO
3	9	08nov1978	Female	HMO
4	16	18may1973	Male	PPO

How Stata Behaves

- When combining datasets, Stata adds one file at a time.
  - The general pattern: open a dataset, add a dataset, check the results, add a dataset, check the results, etc.
- Stata Terminology:
  - The data which are in memory already are called the **master data**.
  - The data which are being added are called the **using data**.

9.2.2 Appending Datasets

Putting All the Visits Together:Visualization



	id	visit	illness	insurance
1	9	26dec2006	Cold	Major Med
2	4	09jan2007	Sore Throat	HMO
3	1	10jan2007	Pneu	.
4	25	02feb2007	Cold	PPO
5	4	05feb2007	Sore Throat	.
6	25	21apr2007	Cold	PPO
7	9	20may2007	Flu	.

	id	visit	illness	insurance	doctor
1	616	24apr2007	Pneu	HMO	Jones
2	9	30may2007	Sore Throat	HMO	Smith

- Stata appends on variable names
  - The variables can be in any order—here they are ordered for illustration only.

Adding Observations from Another Dataset in Stata

- To add observations to the dataset in use from another dataset which has the same structure, use

  `append using "`*filename*`" [, nolabel]`

  - Storage types get promoted so no data will be lost, e.g. appending a float to a byte will make a float.
  - The `nolabel` option suppresses any value labels defined in the using dataset.
    - Rarely used.
- Use when adding new data to old, for example when new data become available in an ongoing project.

Appending New Visit Info to Old Visit Info

- First open up `old_visit_info` so that it is the master dataset.

```
. use "old_visit_info"
```

- Then append the new info

```
. append using "new_visit_info"

id was byte now int
(label instype already defined)
```

Result

	id	visit	illness	insurance	doctor
1	9	26dec2006	Cold	Major Med	
2	4	09jan2007	Sore Throat	HMO	
3	1	10jan2007	Pneu	.	
4	25	02feb2007	Cold	PPO	
5	4	05feb2007	Sore Throat	.	
6	25	21apr2007	Cold	PPO	
7	9	20may2007	Flu	.	
8	616	24apr2007	Pneu	HMO	Jones
9	9	30may2007	Sore Throat	HMO	Smith

- Just what was expected
- Now save this as `all_visit_info`:

```
. save "all_visit_info"

file all_visit_info.dta saved
```

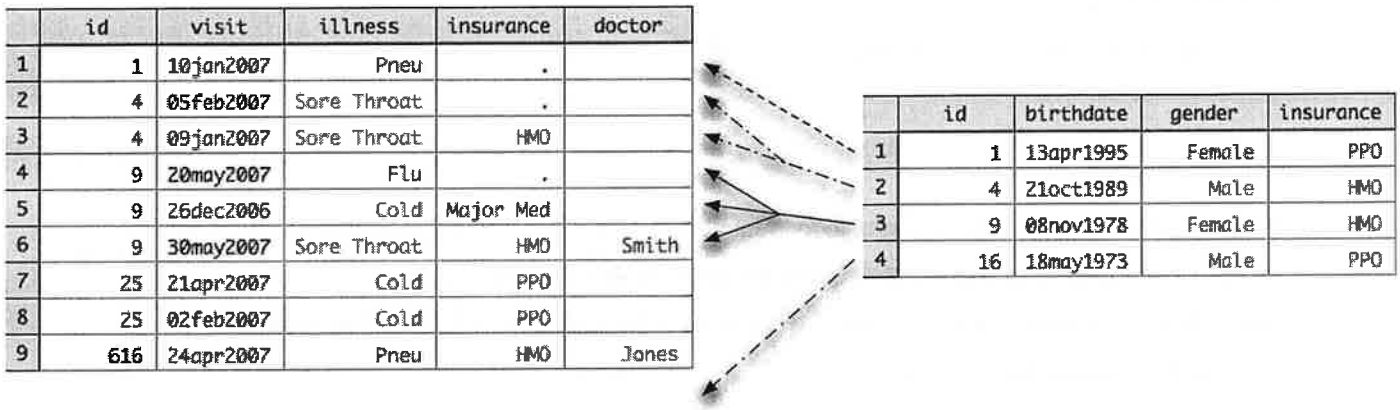
Notes:

- Appending attaches the using data under the master data
- Every variable is in the resulting dataset
- No reordering of the observations takes place
  - If the data in the master dataset were sorted, the resulting dataset will not be sorted anymore
- If you want to keep track of which observations come from which datasets, you need to do this yourself

---

9.2.3 Merging Datasets

Adding the Variables in the Personal Info: Visualization of Desired Behavior



	id	visit	illness	insurance	doctor
1	1	10jan2007	Pneu	.	
2	4	05feb2007	Sore Throat	.	
3	4	09jan2007	Sore Throat	HMO	
4	9	20may2007	Flu	.	
5	9	26dec2006	Cold	Major Med	
6	9	30may2007	Sore Throat	HMO	Smith
7	25	21apr2007	Cold	PPO	
8	25	02feb2007	Cold	PPO	
9	616	24apr2007	Pneu	HMO	Jones

	id	birthdate	gender	insurance
1	1	13apr1995	Female	PPO
2	4	21oct1989	Male	HMO
3	9	08nov1978	Female	HMO
4	16	18may1973	Male	PPO

- Want to match up the information for each patient with that patient's visits.
  - Can be done using the `id` variable, which is in both datasets
- A nuisance: the `insurance` variable appears in both datasets

The Desired Result—Ignoring Insurance

	id	visit	illness	insurance	doctor	birthdate	gender
1	1	10jan2007	Pneu			13apr1995	Female
2	4	05feb2007	Sore Throat			21oct1989	Male
3	4	09jan2007	Sore Throat			21oct1989	Male
4	9	20may2007	Flu			08nov1978	Female
5	9	26dec2006	Cold			08nov1978	Female
6	9	30may2007	Sore Throat		Smith	08nov1978	Female
7	25	21apr2007	Cold			.	
8	25	02feb2007	Cold			.	
9	616	24apr2007	Pneu		Jones	.	
10	16	.				18may1973	Male

Why Can These Datasets Be Joined?

- It is clear that the `id` variable should to be used to join the two files together.
- It is less clear why using `id` will work well, yielding a useful dataset
  - Things work well, because the `id` variable uniquely identifies the individual observations in one of the two files (`patient_info`).
- If no such information were available, we could not join the files properly, because there could be many ways to match observations
  - This means that `id` is a very important variable

Terminology: Primary Key

- A *varlist* which allows identification of individual observations in a dataset is called a *primary key*
- So... we'll be joining the files together using the primary key from the `patient_info` file.
- Aside: This is not needed for our example, but what is the primary key for the visit information?
  - It is the variable list `id visit`

Adding Variables from Another Dataset in Stata

- Adding extra variables from another dataset is done using the `merge` command:
  - `merge` [*keyvarlist*] `using "`*filename*`"` [`,` *options*]
  - Important! Note that the *keyvarlist* is optional, but (as we'll see) it is nearly always needed for a proper merge!
    - Stata's syntax diagrams say varlist, but keyvarlist is clearer!
- Merging without a *keyvarlist* is rare and generally causes mayhem!
  - Stata uses the observation numbers in each dataset as the key

---

Vitally Important `merge` Options

- If a *keyvarlist* is used,
  - Either both datasets need to be sorted by it first, or the `sort` option must be specified.
- You should **always** use either the `uniqmaster` or `uniqusing` option
  - The one you use depends on whose primary key is used
  - If you do not use either option,
    - If you use the `sort` option, Stata assumes the *keyvarlist* is the primary key for **both** files...which is uncommon
    - If you do not use the `sort` option, Stata will happily merge the files even if the *keyvarlist* is not the primary key for either file. This makes a mess!

Properly Adding the Patient Info

- The `all_visit_info` dataset should be in use right now.
  - If it is not, make sure it is.
  - If you are behind, try

```
. use "all_visit_info_cp"
```

- Note (once again) that `id` is the primary key for the `patient_info` dataset.
- Merge in the `personal_info` file:

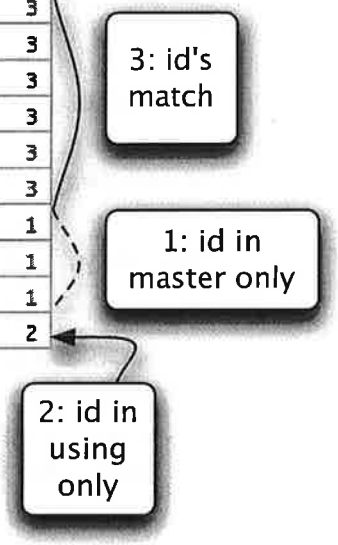
```
. merge id using "patient_info", uniqusing sort
```

```
variable id does not uniquely identify observations in the master data
(label instype already defined)
```

- This does what was pictured earlier, plus a little.

Result

	id	visit	illness	insurance	doctor	birthdate	gender	_merge
1	1	10jan2007	Pneu	.		13apr1995	Female	3
2	4	05feb2007	Sore Throat	.		21oct1989	Male	3
3	4	09jan2007	Sore Throat	HMO		21oct1989	Male	3
4	9	30may2007	Sore Throat	HMO	Smith	08nov1978	Female	3
5	9	26dec2006	Cold	Major Med		08nov1978	Female	3
6	9	20may2007	Flu	.		08nov1978	Female	3
7	25	02feb2007	Cold	PPO		.		1
8	25	21apr2007	Cold	PPO		.		1
9	616	24apr2007	Pneu	HMO	Jones	.		1
10	16	.		PPO		18may1973	Male	2



Using `merge` with a *keyvarlist*: What Stata Does

- Stata compares the *keyvarlist* for each observation in the master and using datasets.
- Whenever the values of the *keyvarlist* match in the master and the using datasets, Stata makes an observation containing all the variables from the two datasets.
- No observations are dropped, so observations for which there are no matches have missing values for the variables from the other dataset.

How `merge` Handles Extra Variables with the Same Names

- When no options are specified, Stata keeps the value from the master dataset, whether it is missing or not
  - The master dataset is the authority
  - A not-quite-exception: If the primary key was found in the using dataset only, its data are brought in for the dupliated variables

Changing the Above

- If the `update` option is specified, Stata will replace missing values in the master dataset with values from the using dataset

- If the `update replace` option is used, Stata replaces values in the master dataset with non-missing values from the using dataset.
  - In this case, the using dataset is almost the authority
    - It won't replace non-missing with missing, however

Trying Out the `update` Option

- To rerun the `merge` command, first open your `all_visit_info` file.

```
. clear
. use "all_visit_info"
```

- Rerun the command with the added `update` option

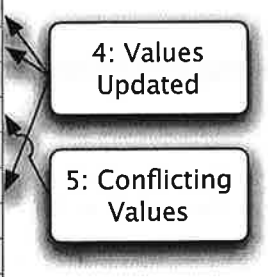
An Example of the `update` Option

- Clear the data out and open the all_visit_info file
- Then run the `merge` command with the `update` option:

```
. merge id using "patient_info", uniqusing sort update

variable id does not uniquely identify observations in the master data
(label instype already defined)
```

Result

	id	visit	illness	insurance	doctor	birthdate	gender	_merge
1	1	10jan2007	Pneu	PPO		13apr1995	Female	4
2	4	05feb2007	Sore Throat	HMO		21oct1989	Male	4
3	4	09jan2007	Sore Throat	HMO		21oct1989	Male	3
4	9	26dec2006	Cold	Major Med		08nov1978	Female	5
5	9	30may2007	Sore Throat	HMO	Smith	08nov1978	Female	3
6	9	20may2007	Flu	HMO		08nov1978	Female	4
7	25	02feb2007	Cold	PPO		.		1
8	25	21apr2007	Cold	PPO		.		1
9	616	24apr2007	Pneu	HMO	Jones	.		1
10	16	.		PPO		18may1973	Male	2



General Results of `merge`

- The `merge` command generates a new variable named `_merge` which contains:
  - a 1 if the key was found only in the master data
  - a 2 if the key was found only in the using data
  - a 3 if the key was found in both datasets (a match)
- When using `update` or `update replace`,
  - a 4 if the keys matched and a missing value in the master dataset replaced with a value from the using dataset.
  - a 5 if the keys matched, and there were unequal non-missing values for a repeated variable in the master and using datasets.

Learning from `_merge`

- The `_merge` results should always be checked to see how well the *keyvarlist* matched!

```
. tab _merge
```

_merge	Freq.	Percent	Cum.
1	3	30.00	30.00
2	1	10.00	40.00
3	2	20.00	60.00
4	3	30.00	90.00
5	1	10.00	100.00
Total	10	100.00	

- Take a careful look at the observations which could not be joined, i.e. if `_merge`< 3,
  - If there are not many, use the browser.
  - If there are many,
    - save the dataset under a new name
    - keep just those observations for which _merge<3
    - run a `codebook`

Finding Data Errors with `_merge`

- You should think about the real-world meanings of `_merge == 1` and `_merge == 2`
- Here, `_merge == 1` whenever there is information about a person for a visit, but there is no information about the person in the patient files
  - This points to problems in data collection
- If `_merge == 2`, there is information about the person, but no visits
  - This should not be worrisome

When Done with `_merge`

- When done with the `_merge` variable, either rename and label it, or drop it. Otherwise, it will likely interfere with subsequent merges.
  - You can give the `_merge` variable a name with the `_merge` option, too.
- Do this here, then save the dataset

```
. drop _merge
. save "all_info"

file all_info.dta saved
```

`merge` Notes—Which Dataset as Master?

- If there are like-named variables, pick the master by what values the final dataset should contain
- If there are no like-named variables
  - `merge` runs slightly faster if the longer of the two datasets is in memory
    - "Longer" means "more observations"
  - This typically happens if the *keyvarlist* is the primary key for the `using` dataset
  - So... the way we did it was the faster way for large datasets

---

`merge` Notes—Terminology

- Database people use the term joining tables, rather than merging datasets
- If the *keyvarlist* defines a primary key in both files, then the merge is called a one-to-one merge
  - This differs from Stata's terminology, but matches that of the relational database world.
- If the keyvarlist defines a primary key in just one file, then the merge is a one-to-many or many-to-one merge
  - Ours was a many-to-one merge
- If the keyvarlist does not define a primary key in either file, then the merge is called a *gawdawful mess.*

Using `merge` with no *keyvarlist*

- When used without the *keyvarlist*, merge puts the datasets together side by side.
  - Each observation in the using dataset is glued to the corresponding observation in the master dataset.
    - Imagine using the observation number as the primary key.
  - This is a rare usage, because it assumes that the observation order is the same in both datasets! You probably don't want to do this!
  - This action is what Stata calls a one-to-one merge.
- Do **not** do this! Ever!

`merge` warnings

- Be careful when merging datasets!
  - Forgetting the key will cause great dismay.
    - Can check for this by looking to see if the `_merge` variable contains only 2's and 3's or only 1's and 3's.
  - If the key is not unique in either of the files, then the result is usually worthless and unpredictable. The error is hard to track down.

9.3 Conclusion

9.3.1 Conclusion

Summary of the Preceding

- Use `append` to add observations.
- Use `merge` to add variables.
  - Be sure there is a primary key which can be used to merge the files.
  - Always use a `uniq...` option
    - Always means always
  - Investigate the `_merge` variable when finished.

Finishing Up

- Close your log file

Lesson 10

Reshaping Data

10.1 Introduction

10.1.1 Goals

Reshaping Data

- Learning about ways to store repeated-measures data
- Changing datasets from wide format to long format and back again

---

10.1.2 Getting Started

Getting Ready

- Create a folder called `reshaping`
- Start a log called `reshaping`
- Download the `reshaping` package from the website.

10.2 Reshaping Data

10.2.1 Reshaping Data

Repeated Measures Data

- Often data are collected multiple times on a group of individuals. These data could be stored two ways:
  - One observation per individual, with variables holding the multiple repeated data.

    - This is called a *wide* dataset.
  - One observation per collection and individual, with multiple observations for each individual.
    - This is called a *long* dataset.

- Switching a dataset from wide to long or long to wide is something both useful and common
  - Wide datasets are in many ways more natural
  - Long datasets are common in relational databases and are more useful for data analysis

---

Our Example

- Open up the `wide` dataset

```
. use wide
```

  - The name matches the format by design, not because of need

- Browse the dataset

Example Wide Dataset

	id	gender	birthdt	birthwt	visdt1	wt1	visdt2	wt2	visdt3	wt3
1	101	Male	26jan1998	1766	29jan1998	1823	08feb1998	2011	23feb1998	2293
2	102	Female	07jan1998	3301	09jan1998	3338	.	.	.	.
3	103	Female	17jan1998	1454	22jan1998	1549	09feb1998	1892	.	.
4	104	Female	08jan1998	3139	16jan1998	3298	18jan1998	3338	20jan1998	3377
5	105	Female	29jan1998	4133	07feb1998	4306	21feb1998	4575	05mar1998	4805

```
Contains data from wide.dta
  obs:             5
 vars:            10                          1 Dec 1998 11:46
 size:           110 (100.0% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
id              byte   %8.0g                  ID
gender          byte   %8.0g       gender     Gender
birthdt         int    %d                     Birth Date
birthwt         int    %9.0g                  Birth Weight in g
visdt1          int    %d                     First Visit Date
wt1             int    %9.0g                  First Visit Weight
visdt2          int    %d                     Second Visit Date
wt2             int    %9.0g                  Second Visit Weight
visdt3          int    %d                     Third Visit Date
wt3             int    %9.0g                  Third Visit Weight
-------------------------------------------------------------------------------
Sorted by:
```

Characteristics of the Wide Dataset

- Each individual represents one observation
- There is a primary key
  - In this example it is made up of the single `id` variable.
- There are repeating variables, whose names consist of a prefix and a numerical suffix.
  - In our example: `wt1`, `wt2`, `wt3`, and `visdt1`, `visdt2`, and `visdt3`
  - There can be many different repeating variables
  - The data in variables with the same numerical suffix belong together
- There can be other non-repeating variables

What Would the Long Dataset In the Example Be?

- Would like to have one observation for each visit, rather than for each person.
  - Need a variable to keep track of the visit number.
    - Note that this variable doesn't exist yet.
  - Would like a single variable for all visit dates
    - Have this information in multiple variables already
    - Each variable name is made up of a prefix (`visdt`) and suffix (*the visit number*).
  - Would like a single variable for the visit weight.
    - Have this information in multiple variables already
    - Each variable name is made up of a prefix (`wt`) and suffix (*the visit number*)

What Could the Long Dataset Look Like?

	id	visit	gender	birthdt	birthwt	visdt	wt
1	101	1	Male	26jan1998	1766	29jan1998	1823
2	101	2	Male	26jan1998	1766	08feb1998	2011
3	101	3	Male	26jan1998	1766	23feb1998	2293
4	102	1	Female	07jan1998	3301	09jan1998	3338
5	102	2	Female	07jan1998	3301	.	.
6	102	3	Female	07jan1998	3301	.	.
7	103	1	Female	17jan1998	1454	22jan1998	1549
8	103	2	Female	17jan1998	1454	09feb1998	1892
9	103	3	Female	17jan1998	1454	.	.
10	104	1	Female	08jan1998	3139	16jan1998	3298
11	104	2	Female	08jan1998	3139	18jan1998	3338
12	104	3	Female	08jan1998	3139	20jan1998	3377
13	105	1	Female	29jan1998	4133	07feb1998	4306
14	105	2	Female	29jan1998	4133	21feb1998	4575
15	105	3	Female	29jan1998	4133	05mar1998	4805

How to get Stata to Switch from Wide to Long

- Stata has the `reshape` command to switch between the two types of datasets
- To get Stata to switch the type of dataset, we first must tell Stata exactly what is needed from the previous slide

---

The Basic Syntax for `reshape`

- Going wide to long:
  - `reshape long` *stubnames*, `i(`*varlist*`) [j(`*varname*`)]`
- Going long to wide:
  - Same syntax, but with `wide` instead of `long`
- `wide` and `long` refer to the final shape of the dataset

- The *stubnames* in the standard part refers to the prefixes of the repeating variables.
- The *varlist* in the `i()` option refers the primary key in the wide dataset.
- The `j()` option is the variable holding the repeat number in the long dataset.

Identifying the Parts

- In the example here,
  - The *stubnames* for the repeated variables are `wt` and `visdt`
  - The key is the `id`
  - We could use `visit` for the visit number.
    - If the `j()` option is not specified, a variable named `_j` is created
    - This is both ugly and conveys little meaning to the average Jane

---

The Command for the Example

- This is what will work here to go to long format:

```
. reshape long wt visdt, i(id) j(visitnum)

(note: j = 1 2 3)

Data                               wide   ->   long
-----------------------------------------------------------------------------
Number of obs.                        5   ->      15
Number of variables                  10   ->       7
j variable (3 values)                     ->   visitnum
xij variables:
                             wt1 wt2 wt3   ->   wt
                   visdt1 visdt2 visdt3    ->   visdt
-----------------------------------------------------------------------------
```

- Browse the resulting dataset
  - By a quirk of nature, it is the long dataset pictured above

---

Notes

- Stata gives some good info about the transformation
  - It tells what possible value there are for the `visitnum`
  - It gives some pretty good documentation about what happens to the variables.
- If there are n different repeat values, going from wide to long will result in n observations for every value of the key
  - Deleting the extras are up to the user

Switching Back In Midstream

- After reshaping, Stata remembers the information needed to transform back and forth between the two formats
- To switch back to the wide format, simply type

```
. reshape wide

(note: j = 1 2 3)

Data                               long   ->   wide
-----------------------------------------------------------------------------
Number of obs.                       15   ->      5
Number of variables                   7   ->     10
j variable (3 values)          visitnum   ->   (dropped)
xij variables:
                                      wt   ->   wt1 wt2 wt3
                                   visdt   ->   visdt1 visdt2 visdt3
-----------------------------------------------------------------------------
```

- You could go back to long again, also

```
. reshape long

(note: j = 1 2 3)

Data                               wide   ->   long
-----------------------------------------------------------------------------
Number of obs.                        5   ->     15
Number of variables                  10   ->      7
j variable (3 values)                     ->   visitnum
xij variables:
                           wt1 wt2 wt3    ->   wt
                  visdt1 visdt2 visdt3    ->   visdt
-----------------------------------------------------------------------------
```

- To see what Stata tracks after using a `reshape` command, type

```
. reshape query

  (data is long)
+-----------------------------------------------------------------------------+
| Xij                           | Command/contents                            |
|-------------------------------+---------------------------------------------|
| Subscript i,j definitions:    |                                             |
|  group id variable(s)         | reshape i id                                |
|  within-group variable        | reshape j visitnum                          |
|   and its range               |                                             |
|                               |                                             |
| Variable X definitions:       |                                             |
|  varying within group         | reshape xij wt visdt                        |
|  constant within group (opt)  | reshape xi  <varlist>                       |
+-----------------------------------------------------------------------------+
Optionally type "reshape xi" to define variables that are constant within i.
Type "reshape wide" to convert the data to wide form.
```

- Note: Variable labels get lost when switching back and forth

A Small Note About the Example

- The `birthwt` and `birthdt` could be handled far better if they were named `wt0` and `visdt0`.
  - This would allow them to become the 'base' observation.
  - This makes the use of survival time-style data analyses far easier.
- Let's do this, while the dataset is wide

```
. rename birthdt visdt0
. rename birthwt wt0
. reshape long

(note: j = 0 1 2 3)

Data                               wide   ->   long
-----------------------------------------------------------------------------
Number of obs.                        5   ->      20
Number of variables                  10   ->       5
j variable (4 values)                     ->   visitnum
xij variables:
                          wt0 wt1 ... wt3   ->   wt
                 visdt0 visdt1 ... visdt3   ->   visdt
-----------------------------------------------------------------------------
```

Some Worthless Observations (in the Data)

- Notice that there are three observations which have missing values for the important variables.
- Drop them

```
. drop if missing(visdt)
```

- You can still make the dataset wide

```
. reshape wide
```

Going Long to Wide from Scratch

- Clear out your dataset, and open the `long` dataset which you downloaded

```
. clear
. use long
```

- Take a look at it
  - It is the long dataset from before, with the missing visits dropped
- To go to wide, use the same logic as before
  - `visdt` and `wt` are prefixes

  - `visitnum` measures repetitions within individual
  - `id` identifies the individuals

- The command will be as before with one word changed:

```
. reshape wide wt visdt, i(id) j(visitnum)

(note: j = 0 1 2 3)

Data                               long   ->   wide
-----------------------------------------------------------------------------
Number of obs.                       17   ->       5
Number of variables                   5   ->      10
j variable (4 values)          visitnum   ->   (dropped)
xij variables:
                                      wt   ->   wt0 wt1 ... wt3
                                   visdt   ->   visdt0 visdt1 ... visdt3
-----------------------------------------------------------------------------
```

Long to Wide Notes

- Stata checks to see that the variables not specified in any of the varlists are constant when switching from long to wide.
- If there were any which changed value, Stata would complain and refuse to reshape

---

Advanced Usage

- The `reshape` command allows changing of the varlists by separate commands, such as

```
reshape j varname [values]
```

- If you are curious, look in the manuals, or on the on-line help. What we've seen will cover most everything.

10.3 Conclusion

10.3.1 Conclusion

Conclusion

- Stata has a nice tool for changing dataset formats
  - This is one of Stata's strengths—it recognizes different types of general dataset types

---

Finishing Up

- Close your log file

Lesson 11

Estimation and Postestimation

11.1 Introduction

11.1.1 Goals

Goals

- Knowing basic rules of how Stata does estimation
- Working with dummy variables and interactions
- Alternative standard errors
- Reusing estimation results
- Using common post-estimation commands
- Fitting model from a training dataset to a testing dataset

11.1.2 Getting Started

Getting Ready

- Create a folder
- Start a log
- Open the Hosmer and Lemeshow dataset

```
. webuse lbw

(Hosmer & Lemeshow data)
```

- Investigate the dataset, if you like

11.2 Estimation

11.2.1 Basic Estimation

Commands and General Syntax

- Look at the help: `help estimation commands`
- Stata's estimation models follow the general syntax

  command depvar indepvars [*if*][*in*][*weight*][, `options`]

- This is true whether using using linear regression (via `regress`), logistic regression (via `logistic` or `logit`), poisson regression (via `poisson`), or any other univariate model

---

Two Simple Examples

- Fitting a linear birth weight model

```
. regress bwt age smoke ui
```

```
      Source |       SS       df       MS              Number of obs =     189
-------------+------------------------------           F(  3,   185) =    7.92
       Model |  11367926.2     3  3789308.73            Prob > F      =  0.0001
    Residual |  88547372.4   185  478634.445            R-squared     =  0.1138
-------------+------------------------------           Adj R-squared =  0.0994
       Total |  99915298.6   188  531464.354            Root MSE      =  691.83

------------------------------------------------------------------------------
         bwt |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         age |   8.503649   9.557205     0.89   0.375    -10.35147    27.35877
       smoke |  -253.7586   103.3842    -2.45   0.015    -457.7222   -49.79511
          ui |  -548.5662    142.308    -3.85   0.000    -829.3215    -267.811
       _cons |   2927.301   235.0232    12.46   0.000     2463.631    3390.972
------------------------------------------------------------------------------
```

- Fitting a logistic probability of a low birth weight

```
. logit low age smoke ui
```

```
Iteration 0:   log likelihood =   -117.336
Iteration 1:   log likelihood =  -111.6356
Iteration 2:   log likelihood = -111.58057
Iteration 3:   log likelihood = -111.58056

Logistic regression                                     Number of obs   =        189
                                                        LR chi2(3)      =      11.51
                                                        Prob > chi2     =     0.0093
Log likelihood = -111.58056                             Pseudo R2       =     0.0491

------------------------------------------------------------------------------
```

```
         low |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         age |  -.0454698   .0323318    -1.41   0.160     -.108839     .0178994
       smoke |    .663463   .3260922     2.03   0.042      .024334     1.302592
          ui |   .8731237   .4271019     2.04   0.041     .0360193     1.710228
       _cons |  -.168955    .7755271    -0.22   0.828     -1.68896      1.35105
------------------------------------------------------------------------------
```

- For seeing odds ratios, either use the `logistic` command or use the `or` option

```
. logit low age smoke ui, or

Iteration 0:   log likelihood =   -117.336
Iteration 1:   log likelihood =  -111.6356
Iteration 2:   log likelihood = -111.58057
Iteration 3:   log likelihood = -111.58056

Logistic regression                                     Number of obs   =        189
                                                        LR chi2(3)      =      11.51
                                                        Prob > chi2     =     0.0093
Log likelihood = -111.58056                             Pseudo R2       =     0.0491

------------------------------------------------------------------------------
         low | Odds Ratio   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         age |   .9555485   .0308946    -1.41   0.160     .8968748    1.018061
       smoke |   1.941504   .6331095     2.03   0.042     1.024632     3.67882
          ui |   2.394378   1.022644     2.04   0.041     1.036676    5.530222
------------------------------------------------------------------------------
```

Replaying Results

- To see the results of the most recent estimation command, either
  - Type the estimation command again with no arguments, or

```
. estimates
------------------------------------------------------------------------------
active results
------------------------------------------------------------------------------

Logistic regression                                     Number of obs   =        189
                                                        LR chi2(3)      =      11.51
                                                        Prob > chi2     =     0.0093
Log likelihood = -111.58056                             Pseudo R2       =     0.0491

------------------------------------------------------------------------------
         low |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         age |  -.0454698   .0323318    -1.41   0.160     -.108839     .0178994
       smoke |    .663463   .3260922     2.03   0.042      .024334     1.302592
          ui |   .8731237   .4271019     2.04   0.041     .0360193     1.710228
       _cons |  -.168955    .7755271    -0.22   0.828     -1.68896      1.35105
------------------------------------------------------------------------------
```

- The results are given without needing to recompute them
  - This is important if the estimation was time-intensive

11.2.2 Categorical Variables and Interactions

Working with Categorical Variables

- Instead of generating new variables for each level of a categorical variable, the `xi:` prefix can be used.
- Put a `i.` in front of each categorical variable name.
  - Stata will generate the required dummy (indicator) variables
  - The variable names start with stliteral_I, by default
- Example:

```
. xi: logit low age smoke ui i.race, or

i.race            _Irace_1-3          (naturally coded; _Irace_1 omitted)

Iteration 0:   log likelihood =   -117.336
Iteration 1:   log likelihood = -107.75636
Iteration 2:   log likelihood = -107.53184
Iteration 3:   log likelihood = -107.53115

Logistic regression                               Number of obs   =        189
                                                  LR chi2(5)      =      19.61
                                                  Prob > chi2     =     0.0015
Log likelihood = -107.53115                       Pseudo R2       =     0.0836

------------------------------------------------------------------------------
         low | Odds Ratio   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         age |   .9703961   .0327444    -0.89   0.373     .9082945    1.036744
       smoke |   2.897143   1.090561     2.83   0.005     1.385336    6.058774
          ui |   2.352306   1.025405     1.96   0.050     1.001019     5.52771
    _Irace_2 |   2.864375   1.416766     2.13   0.033     1.086455    7.551755
    _Irace_3 |   2.791546   1.150491     2.49   0.013     1.244617    6.261144
------------------------------------------------------------------------------
```

Setting `xi:`'s Base Category to a Value

- By default, the smallest-valued category is the base category.
  - In this case, `race==1`, or white
- The following odd-looking command will change the base category to a specific value:

`char` *varname* `[omit]` *value*

  - This uses Stata's `chars`, which are used to store arbitrary information tied to a variable

- Here, if we wanted the base category to be 2 and rerun the command

```
. char race[omit] 2
. xi: logit low age smoke ui i.race, or

i.race            _Irace_1-3          (naturally coded; _Irace_2 omitted)

Iteration 0:   log likelihood =   -117.336
Iteration 1:   log likelihood = -107.75636
Iteration 2:   log likelihood = -107.53184
Iteration 3:   log likelihood = -107.53115
Iteration 4:   log likelihood = -107.53115

Logistic regression                                     Number of obs   =        189
                                                        LR chi2(5)      =      19.61
                                                        Prob > chi2     =     0.0015
Log likelihood = -107.53115                             Pseudo R2       =     0.0836

------------------------------------------------------------------------------
         low | Odds Ratio   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         age |   .9703961   .0327446    -0.89   0.373     .9082941    1.036744
       smoke |   2.897143   1.090569     2.83   0.005     1.385328    6.058807
          ui |   2.352306   1.025408     1.96   0.050     1.001017    5.527723
    _Irace_1 |   .3491163   .1726792    -2.13   0.033     .1324191    .9204279
    _Irace_3 |   .9745742   .4809277    -0.05   0.958     .3704831    2.563666
------------------------------------------------------------------------------
```

- To go back to the default behavior, set the characteristic to nothingness

```
. char race[omit]
```

Setting `xi:`'s Base Category to Most Prevalent

- It is also possible to change the default behavior for the dataset to omit the most prevalent value
- In this case, the "variable" must be given as `_dta` to use the most prevalent category for all variables in the dataset

```
. char _dta[omit] prevalent
```

- Variable-level settings override dataset-level settings

Working with Interactions

- The `xi:` prefix is used for interactions of a categorical variable with either a categorical or continuous variable
- The interaction term is expressed by putting `*` between terms to include main effects, or `|` to include only the interaction terms

- Example

```
. xi: logit low age ui i.race*smoke, or

i.race            _Irace_1-3          (naturally coded; _Irace_1 omitted)
i.race*smoke      _IracXsmoke_#       (coded as above)

Iteration 0:   log likelihood =   -117.336
Iteration 1:   log likelihood = -106.39678
Iteration 2:   log likelihood = -105.80902
Iteration 3:   log likelihood = -105.79483
Iteration 4:   log likelihood = -105.79482

Logistic regression                                     Number of obs   =        189
                                                        LR chi2(7)      =      23.08
                                                        Prob > chi2     =     0.0016
Log likelihood = -105.79482                             Pseudo R2       =     0.0984

------------------------------------------------------------------------------
         low | Odds Ratio   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         age |   .9741864   .0343035    -0.74   0.458     .9092204    1.043794
          ui |   2.546774   1.125319     2.12   0.034     1.071218     6.05484
    _Irace_2 |   3.610687   2.834837     1.64   0.102      .774981    16.82242
    _Irace_3 |   5.069789   3.093346     2.66   0.008     1.533299    16.76304
       smoke |   5.018923   3.073528     2.63   0.008      1.51128    16.66772
_IracXsmok~2 |   .8922063   .9600113    -0.11   0.916     .1082867    7.351156
_IracXsmok~3 |   .2091348   .1900177    -1.72   0.085     .0352398    1.241136
------------------------------------------------------------------------------
```

Other Notes

- To fit one model for each level of a categorical variable, use a `by:` prefix command

```
. by race, sort: logit low age smoke ui, or

------------------------------------------------------------------------------
-> race = white

Iteration 0:   log likelihood =  -52.85752
Iteration 1:   log likelihood = -47.382245
Iteration 2:   log likelihood = -47.087952
Iteration 3:   log likelihood = -47.084654
Iteration 4:   log likelihood = -47.084653

Logistic regression                                     Number of obs   =         96
                                                        LR chi2(3)      =      11.55
                                                        Prob > chi2     =     0.0091
Log likelihood = -47.084653                             Pseudo R2       =     0.1092

------------------------------------------------------------------------------
         low | Odds Ratio   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
```

```
         age |   .9885006   .0501096    -0.23   0.820      .895009    1.091758
       smoke |    5.32098   3.299863     2.70   0.007     1.57801    17.94211
          ui |     1.8201   1.231703     0.88   0.376     .4831343    6.856816
------------------------------------------------------------------------------

------------------------------------------------------------------------------
-> race = black

Iteration 0:   log likelihood = -17.712909
Iteration 1:   log likelihood = -15.650683
Iteration 2:   log likelihood = -15.636832
Iteration 3:   log likelihood = -15.636822

Logistic regression                               Number of obs   =         26
                                                  LR chi2(3)      =       4.15
                                                  Prob > chi2     =     0.2455
Log likelihood = -15.636822                       Pseudo R2       =     0.1172

------------------------------------------------------------------------------
         low | Odds Ratio   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         age |   .9791734    .091251    -0.23   0.821     .8157076    1.175397
       smoke |   5.537469   5.720873     1.66   0.098     .7310001    41.94741
          ui |   7.110651   10.11412     1.38   0.168     .4376924     115.518
------------------------------------------------------------------------------

------------------------------------------------------------------------------
-> race = other

Iteration 0:   log likelihood = -44.260386
Iteration 1:   log likelihood = -42.416092
Iteration 2:   log likelihood = -42.410156
Iteration 3:   log likelihood = -42.410156

Logistic regression                               Number of obs   =         67
                                                  LR chi2(3)      =       3.70
                                                  Prob > chi2     =     0.2957
Log likelihood = -42.410156                       Pseudo R2       =     0.0418

------------------------------------------------------------------------------
         low | Odds Ratio   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         age |   .9426665   .0556882    -1.00   0.318     .8396015    1.058383
       smoke |   1.032736   .7108159     0.05   0.963      .267991     3.97977
          ui |   2.830441   1.895166     1.55   0.120     .7619368    10.51452
------------------------------------------------------------------------------
```

- The `sort` option in the `by` prefix command is needed if the dataset is not sorted by `race`
  - If it is, including the sort option doesn't hurt
- There is an equivalent to this: `bysort`

```
. bysort race: logit low age smoke ui, or

------------------------------------------------------------------------------
-> race = white
```

```
Iteration 0:   log likelihood =  -52.85752
Iteration 1:   log likelihood = -47.382245
Iteration 2:   log likelihood = -47.087952
Iteration 3:   log likelihood = -47.084654
Iteration 4:   log likelihood = -47.084653

Logistic regression                               Number of obs   =         96
                                                  LR chi2(3)      =      11.55
                                                  Prob > chi2     =     0.0091
Log likelihood = -47.084653                       Pseudo R2       =     0.1092

------------------------------------------------------------------------------
         low | Odds Ratio   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         age |   .9885006   .0501096    -0.23   0.820      .895009    1.091758
       smoke |    5.32098   3.299863     2.70   0.007      1.57801    17.94211
          ui |     1.8201   1.231703     0.88   0.376     .4831343    6.856816
------------------------------------------------------------------------------

-------------------------------------------------------------------------------
-> race = black

Iteration 0:   log likelihood = -17.712909
Iteration 1:   log likelihood = -15.650683
Iteration 2:   log likelihood = -15.636832
Iteration 3:   log likelihood = -15.636822

Logistic regression                               Number of obs   =         26
                                                  LR chi2(3)      =       4.15
                                                  Prob > chi2     =     0.2455
Log likelihood = -15.636822                       Pseudo R2       =     0.1172

------------------------------------------------------------------------------
         low | Odds Ratio   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         age |   .9791734    .091251    -0.23   0.821     .8157076    1.175397
       smoke |   5.537469   5.720873     1.66   0.098     .7310001    41.94741
          ui |   7.110651   10.11412     1.38   0.168     .4376924     115.518
------------------------------------------------------------------------------

-------------------------------------------------------------------------------
-> race = other

Iteration 0:   log likelihood = -44.260386
Iteration 1:   log likelihood = -42.416092
Iteration 2:   log likelihood = -42.410156
Iteration 3:   log likelihood = -42.410156

Logistic regression                               Number of obs   =         67
                                                  LR chi2(3)      =       3.70
                                                  Prob > chi2     =     0.2957
Log likelihood = -42.410156                       Pseudo R2       =     0.0418

------------------------------------------------------------------------------
```

```
         low | Odds Ratio   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         age |   .9426665   .0556882    -1.00   0.318     .8396015    1.058383
       smoke |   1.032736   .7108159     0.05   0.963      .267991     3.97977
          ui |   2.830441   1.895166     1.55   0.120     .7619368    10.51452
------------------------------------------------------------------------------
```

What Observations Get Used?

- Stata uses the most data it can
- All observations for which the variables specified by the model are non-missing are used
- This requires a little thought, because different models might be fit on different sets of observations
- As we'll see, Stata has a tool for keeping track of observations used in estimation commands

11.2.3 Alternate Standard Error Computations

General Syntax

If this looks hard to remember, you can find the various computations on the ***SE/Robust*** tab of the dialog box for the command.

- Most of Stata's estimation commands allow the `vce` option to specify alternate methods for computing the variance-covariance matrix of the estimators
- These VCE estimates can be useful when either the model specification or the model assumptions are suspect

Sandwich Estimators

- Specifying `vce(robust)` yields Huber-White sandwich estimates

```
. logit low age smoke ui, or vce(robust)

Iteration 0:   log pseudolikelihood =   -117.336
Iteration 1:   log pseudolikelihood =  -111.6356
Iteration 2:   log pseudolikelihood = -111.58057
Iteration 3:   log pseudolikelihood = -111.58056

Logistic regression                               Number of obs   =        189
                                                  Wald chi2(3)    =      11.64
                                                  Prob > chi2     =     0.0087
Log pseudolikelihood = -111.58056                 Pseudo R2       =     0.0491

------------------------------------------------------------------------------
             |               Robust
```

```
         low | Odds Ratio   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         age |   .9555485   .0283836    -1.53   0.126     .9015061     1.01283
       smoke |   1.941504   .6399167     2.01   0.044     1.017615     3.704188
          ui |   2.394378   1.047807     2.00   0.046     1.015541     5.645313
------------------------------------------------------------------------------
```

- This estimate of the standard error is more robust against model misspecification

---

Clustered Data

- If data naturally come from clusters, specifying `vce(cluster` *varname*`)` will adjust the standard error estimates
  - This could arise if, for example, observations were clustered within hospitals
- Using this option implies that sandwich estimators are also used

---

Bootstrap Estimates

- Specifying `vce(bootstrap)` will compute bootstrap errrors
  - For replicability, set **both** a unique sort order and the random seed before using!

```
. sort id
. set seed 142857
. logit low age smoke ui, or vce(bootstrap)
(running logit on estimation sample)

Bootstrap replications (50)
----+--- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5

Logistic regression                             Number of obs     =        189
                                                Replications      =         50
                                                Wald chi2(3)      =      10.51
                                                Prob > chi2       =     0.0147
Log likelihood = -111.58056                     Pseudo R2         =     0.0491

------------------------------------------------------------------------------
             |   Observed   Bootstrap                         Normal-based
         low | Odds Ratio   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         age |   .9555485   .0282688    -1.54   0.124     .9017183    1.012592
       smoke |   1.941504   .7311107     1.76   0.078     .9281155     4.06139
          ui |   2.394378   1.264806     1.65   0.098     .8502649    6.742661
------------------------------------------------------------------------------
```

  - There are many options which can be used the option (see `help bootstrap`
- These can be of value when it is thought that the estimates are biased or if there are model violations

---

Jackknife Estimates

- Specifying `vce(jackknife)` will compute the the VCE using the all-but-one jackknife.

```
. logit low age smoke ui, vce(jackknife)

(running logit on estimation sample)

Jackknife replications (189)
----+--- 1 ---+--- 2 ---+--- 3 ---+--- 4 ---+--- 5

Logistic regression                             Number of obs      =       189
                                                Replications       =       189
                                                F(  3,    188)     =      3.64
                                                Prob > F           =    0.0138
Log likelihood = -111.58056                     Pseudo R2          =    0.0491

------------------------------------------------------------------------------
             |              Jackknife
         low |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         age |  -.0454698   .0305978    -1.49   0.139    -.1058288    .0148893
       smoke |    .663463   .3371592     1.97   0.051    -.0016383    1.328564
          ui |   .8731237   .4555211     1.92   0.057    -.0254659    1.771713
       _cons |   -.168955   .7593867    -0.22   0.824    -1.666969    1.329059
------------------------------------------------------------------------------
```

- This is meant to reduce bias

11.2.4 Estimation Results

Estimation Results

- Estimation commands produce estimation-specific results which can be reused
- These can be seen by using the command

```
. ereturn list

scalars:
         e(N_misreps) =  0
            e(N_reps) =  189
              e(df_r) =  188
             e(N_cds) =  0
             e(N_cdf) =  0
              e(r2_p) =  .0490509011521354
                e(ll) =  -111.5805616523853
              e(ll_0) =  -117.3359980966092
              e(k_eq) =  1
             e(k_exp) =  0
            e(k_eexp) =  4
           e(k_extra) =  0
         e(singleton) =  .
```

```
                  e(N) =  189
               e(df_m) =  3
                  e(F) =  3.642807403621608
                  e(p) =  .0137730723162002

macros:
            e(cmdline) : "logit low age smoke ui, vce(jackknife)"
                e(cmd) : "logit"
             e(prefix) : "jackknife"
            e(cmdname) : "logit"
          e(nfunction) : "e(N)"
            e(command) : "logit low age smoke ui"
               e(exp4) : "_b[_cons]"
               e(exp3) : "_b[ui]"
               e(exp2) : "_b[smoke]"
               e(exp1) : "_b[age]"
              e(title) : "Logistic regression"
          e(estat_cmd) : "logit_estat"
            e(predict) : "logit_p"
           e(crittype) : "log likelihood"
             e(depvar) : "low"
                e(vce) : "jackknife"
            e(vcetype) : "Jackknife"
        e(properties) : "b V"

matrices:
                  e(b) :  1 x 4
                  e(V) :  4 x 4
              e(rules) :  1 x 4
               e(b_jk) :  1 x 4

functions:
            e(sample)
```

- Estimation commands are *e-class* commands
- The results in the `ereturn list` last until the next estimation command is issued

---

Some Notes about Estimation Results

- The scalars and macros can be used in the same way as those in a `return list`
- There are two new items: matrices and a function.
- The matrices' contents can be listed by `matrix list` *matrix*

```
. matrix list e(b)

e(b)[1,4]
           age       smoke          ui       _cons
y1  -.04546979   .66346302   .87312366  -.16895497
```

  - While there is plenty we can do with matrices, we won't spend time on them now

---

What is `e(sample)`?

- The `e(sample)` function is more immediately useful: it can be used to find which observations were used in an estimation.
- The `e(sample)` function can be used to restrict a command to just those observations which were originally used in the estimation
- Most useful when there are missing data or if the model was fit on a subset of the data using an `if` or `in` clause.

Storing and Saving Estimation Results

- There is more to the estimation results than the e-class results: look at `help estimates`
  - For one: estimation results can be *stored* and `saved`
- Storing the results keeps them for later use in a session
  - This is needed for making tables of estimates and looking at likelihood ratio tests or information criteria
- Saving the results stores them on disk for later retrieval
  - This is useful for saving results across sessions
  - It is good for sharing models with others, as they can read the file
  - It is helpful when fitting very time-intesive models

Estimates Store Example

- Let's run two models and store the results
- We'll then use these results when looking at some post-estimation commands later
- First, a big model

```
. xi: logit low age lwt smoke ui i.race

i.race            _Irace_1-3          (naturally coded; _Irace_1 omitted)

Iteration 0:   log likelihood =   -117.336
Iteration 1:   log likelihood = -106.14242
Iteration 2:   log likelihood = -105.80071
Iteration 3:   log likelihood = -105.79896
Iteration 4:   log likelihood = -105.79896

Logistic regression                                     Number of obs   =        189
                                                        LR chi2(6)      =      23.07
                                                        Prob > chi2     =     0.0008
Log likelihood = -105.79896                             Pseudo R2       =     0.0983
```

```
------------------------------------------------------------------------------
         low |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         age |   -.0197546   .0344038    -0.57   0.566    -.0871848    .0476756
         lwt |   -.0113698   .0064025    -1.78   0.076    -.0239184    .0011788
       smoke |    1.022659   .3833948     2.67   0.008     .2712193    1.774099
          ui |    .7669792   .4402573     1.74   0.081    -.0959094    1.629868
      _Irace_2 |    1.256841   .5165319     2.43   0.015     .2444573    2.269225
      _Irace_3 |    .9294705   .4213411     2.21   0.027     .1036571    1.755284
       _cons |    .0104206   1.135112     0.01   0.993    -2.214357    2.235199
------------------------------------------------------------------------------
```

- Now store the estimates

```
. estimates store low_alsur
```

- Now, a smaller model, taking care to use the same obserivations

```
. xi: logit low smoke ui i.race if e(sample)

i.race            _Irace_1-3          (naturally coded; _Irace_1 omitted)

Iteration 0:   log likelihood =   -117.336
Iteration 1:   log likelihood = -108.10806
Iteration 2:   log likelihood = -107.93433
Iteration 3:   log likelihood = -107.93404

Logistic regression                               Number of obs   =        189
                                                  LR chi2(4)      =      18.80
                                                  Prob > chi2     =     0.0009
Log likelihood = -107.93404                       Pseudo R2       =     0.0801

------------------------------------------------------------------------------
         low |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
       smoke |    1.080361   .3740433     2.89   0.004     .3472495    1.813472
          ui |    .8834084   .4329481     2.04   0.041     .0348457    1.731971
      _Irace_2 |    1.116041   .4907334     2.27   0.023     .1542216    2.077861
      _Irace_3 |    1.072471   .4069078     2.64   0.008     .2749467    1.869996
       _cons |   -1.964536   .3653468    -5.38   0.000    -2.680603   -1.248469
------------------------------------------------------------------------------
```

- Store it, also:

```
. estimates store low_sur
```

Estimates Table Example

- We can make tables of coefficients

```
. estimates table low_alsur low_sur
```

```
-------------------------------------
    Variable | low_alsur     low_sur
-------------+-----------------------
         age |  -.01975459
         lwt |  -.01136981
       smoke |   1.0226593    1.0803609
          ui |   .76697916    .88340839
      _Irace_2 |   1.2568412    1.1160415
      _Irace_3 |   .92947049    1.0724713
       _cons |   .01042063   -1.9645359
-------------------------------------
```

- The model currently in memory can be specified using a period (.)—the following gives the same results

```
. estimates table low_alsur .
```

More about Estimates

- We'll see more shortly
- It is better to treat these in the postestimation command section

---

11.3 Common Postestimation Commands

11.3.1 Common Postestimation Commands

What is Available?

- There are common commands and estimation-specific commands
  - The common commands cover activities which would be used after most any estimation
    - Stata calls these "standard" postestimation commands
  - The specific commands are special to the particular type or class of estimation
- To see what is available for `regress` type

```
help regress postestimation
```

  - The help is organized with the specific commands on top—we will start with the common commands below

---

Simple Summary Stats

- One of the useful quick-and-dirty postestimation commands is `estat`—look at it's help
- Here are the summary statistics for the variables used in the last model fit:

```
. estat summarize

Estimation sample logit                   Number of obs =    189

------------------------------------------------------------------
    Variable |        Mean     Std. Dev.       Min        Max
-------------+----------------------------------------------------
         low |    .3121693     .4646093           0          1
       smoke |    .3915344     .4893898           0          1
          ui |    .1481481     .3561903           0          1
    _Irace_2 |    .1375661     .3453589           0          1
    _Irace_3 |    .3544974     .4796313           0          1
------------------------------------------------------------------
```

- While we don't see it here, `estat` restricts the summary statistics to missing values only

---

11.3.2 Postestimation Tests and Estimation

Testing Hypotheses about Coefficients

- The `test` and `testnl` allow testing hypotheses about the coefficients
  - `test` is for linear combinations (contrasts)
  - `testnl` is for non-linear combinations

Example: Linear Tests

- Fit the following:

```
. xi: logit low age smoke ui i.race, or

i.race            _Irace_1-3          (naturally coded; _Irace_1 omitted)

Iteration 0:   log likelihood =   -117.336
Iteration 1:   log likelihood = -107.75636
Iteration 2:   log likelihood = -107.53184
Iteration 3:   log likelihood = -107.53115

Logistic regression                               Number of obs   =        189
                                                  LR chi2(5)      =      19.61
                                                  Prob > chi2     =     0.0015
Log likelihood = -107.53115                       Pseudo R2       =     0.0836
```

```
------------------------------------------------------------------------------
         low | Odds Ratio   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         age |   .9703961   .0327444    -0.89   0.373     .9082945    1.036744
       smoke |   2.897143   1.090561     2.83   0.005     1.385336    6.058774
          ui |   2.352306   1.025405     1.96   0.050     1.001019     5.52771
    _Irace_2 |   2.864375   1.416766     2.13   0.033     1.086455    7.551755
    _Irace_3 |   2.791546   1.150491     2.49   0.013     1.244617    6.261144
------------------------------------------------------------------------------
```

- Now test to see if the coefficients for 'black' and 'other' are different:

```
. test _Irace_2 == _Irace_3

 ( 1)  _Irace_2 - _Irace_3 = 0

           chi2(  1) =    0.00
         Prob > chi2 =    0.9584
```

- Something silly

```
. test 100*age + 2*smoke == ///
.   smoke - _Irace_2 - _Irace_3

 ( 1)  100 age + smoke + _Irace_2 + _Irace_3 = 0

           chi2(  1) =    0.00
         Prob > chi2 =    0.9697
```

Example: Multiple Tests

- Multiple tests can be run by enclosing the separate tests in parentheses
- If we wanted to simultaneously test coefficients (as above) and the `age` coefficient, we could do this

```
. test (_Irace_2 == _Irace_3) (age)

 ( 1)  _Irace_2 - _Irace_3 = 0
 ( 2)  age = 0

           chi2(  2) =    0.80
         Prob > chi2 =    0.6700
```

  - Specifying a coefficient by itself is equivalent to testing if it is 0

Notes about `test`

- The coefficient is represented by its variable name
- `test` is tolerant of a single = sign
- There can be multiple tests run at once
- The tests can be more complex, as long as they are linear

Example: A Non-linear Test

- Now try the following admittedly silly test

```
. testnl _b[smoke]/_b[ui] == _b[_Irace_2]/_b[_Irace_3]

  (1)  _b[smoke]/_b[ui] = _b[_Irace_2]/_b[_Irace_3]

               chi2(1) =        0.05
           Prob > chi2 =        0.8232
```

- This looks a little more difficult

Notes about `testnl`

- The coefficient is must be represented by its coefficient's name: `_b[`*varname*`]`
- `nltest` is tolerant of a single = sign
- There can be multiple tests run at once

Estimating Linear Combinations of Coefficients

- Linear combinations may be estimated (as well as tested)
- This is done via the `lincom` command
- Similar to `test`, except that we estimate an expression, instead of testing an equation

```
. lincom _Irace_2 - _Irace_3

 ( 1)  _Irace_2 - _Irace_3 = 0

------------------------------------------------------------------------------
         low |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         (1) |   .0257546   .4934739     0.05   0.958    -.9414365    .9929457
------------------------------------------------------------------------------
```

Estimating Non-linear Combinations of Coefficients

- Non-linear combinations may be estimated (as well as tested)
- This is done via the `nlcom` command
- Similar to `testnl`

```
. nlcom _b[smoke]/_b[ui] - _b[_Irace_2]/_b[_Irace_3]

       _nl_1:  _b[smoke]/_b[ui] - _b[_Irace_2]/_b[_Irace_3]

------------------------------------------------------------------------------
         low |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
       _nl_1 |   .2184595   .9777658     0.22   0.823    -1.697926    2.134845
------------------------------------------------------------------------------
```

11.3.3 Predicted Values

Predicted Values

- The `predict` command is used to get various kinds of predicted values after estimation
- The default syntax usage is quite simple:

  `predict` *newvar*

- This generally gives what you would expect
  - Predicted value for `regress`
  - Predicted probability of being 1 for `logit` or `logistic`
- For our last model:

```
. predict prlow

(option pr assumed; Pr(low))
```

- We can see that the predicted probabilities don't vary much, even within smoking category

```
. tab smoke, sum(prlow)

    smoked |
    during |          Summary of Pr(low)
  pregnancy |       Mean   Std. Dev.       Freq.
------------+------------------------------------
          0 |  .25217391  .11708056         115
          1 |   .4054054  .14323547          74
------------+------------------------------------
      Total |  .31216931  .14799384         189
```

Out of Sample Predictions

- The `predict` postestimation command can be used to predict probabilities for out-of-sample observations
  - The observations can even be in another dataset!
- This can be used for what-if predictions

```
. xi: logit low age ui i.race if smoke==0

i.race            _Irace_1-3          (naturally coded; _Irace_1 omitted)

Iteration 0:   log likelihood = -64.941748
Iteration 1:   log likelihood = -57.751229
Iteration 2:   log likelihood =  -57.30044
Iteration 3:   log likelihood = -57.292589
Iteration 4:   log likelihood = -57.292584

Logistic regression                               Number of obs   =        115
                                                  LR chi2(4)      =      15.30
                                                  Prob > chi2     =     0.0041
Log likelihood = -57.292584                       Pseudo R2       =     0.1178

------------------------------------------------------------------------------
         low |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         age |  -.0554218   .0518856    -1.07   0.285    -.1571157    .0462722
          ui |   1.130671   .6160206     1.84   0.066    -.0767076    2.338049
    _Irace_2 |   1.117205   .8099554     1.38   0.168    -.4702785    2.704688
    _Irace_3 |   1.536572   .6193716     2.48   0.013     .3226256    2.750518
       _cons |  -1.041696   1.383604    -0.75   0.452     -3.75351    1.670117
------------------------------------------------------------------------------

. predict prlownon

(option pr assumed; Pr(low))

. tabstat prlownon, stat(mean) by(smoke)

Summary for variables: prlownon
     by categories of: smoke (smoked during pregnancy)

   smoke |      mean
---------+----------
       0 |  .2521739
       1 |   .182451
---------+----------
   Total |   .224875
--------------------
```

Adjusted Values

- This allows what-if predictions by setting covariates to a fixed value

```
. xi: logit low age smoke ui i.race, or

i.race            _Irace_1-3          (naturally coded; _Irace_1 omitted)

Iteration 0:   log likelihood =   -117.336
Iteration 1:   log likelihood = -107.75636
Iteration 2:   log likelihood = -107.53184
Iteration 3:   log likelihood = -107.53115

Logistic regression                                     Number of obs   =        189
                                                        LR chi2(5)      =      19.61
                                                        Prob > chi2     =     0.0015
Log likelihood = -107.53115                             Pseudo R2       =     0.0836

------------------------------------------------------------------------------
         low | Odds Ratio   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         age |   .9703961   .0327444    -0.89   0.373     .9082945    1.036744
       smoke |   2.897143   1.090561     2.83   0.005     1.385336    6.058774
          ui |   2.352306   1.025405     1.96   0.050     1.001019     5.52771
    _Irace_2 |   2.864375   1.416766     2.13   0.033     1.086455    7.551755
    _Irace_3 |   2.791546   1.150491     2.49   0.013     1.244617    6.261144
------------------------------------------------------------------------------

. adjust age = 22, pr

------------------------------------------------------------------------------
     Dependent variable: low     Command: logit
   Variables left as is: smoke, ui, _Irace_2, _Irace_3
 Covariate set to value: age = 22
------------------------------------------------------------------------------

----------------------
      All |         pr
----------+-----------
          |    .299784
----------------------
     Key:  pr  =  Probability

. adjust smoke=0, pr by(race)

------------------------------------------------------------------------------
     Dependent variable: low     Command: logit
   Variables left as is: age, ui, _Irace_2, _Irace_3
 Covariate set to value: smoke = 0
------------------------------------------------------------------------------

----------------------
     race |         pr
----------+-----------
    white |     .13549
    black |     .32403
```

```
       other |     .324736
-------------------------
         Key:  pr  =  Probability
```

- Note that the `pr` option is required for `logit` or similar commands

---

11.3.4 Comparing Models

Comparing Models

- Both nested models and non-nested models my be compared
- Nested models may be tested using likelihood ratio tests
- Both require *storing* the results of one model to compare to the other

---

Nested Models, Algorithm from Scratch

- To run a likelihood ratio test for nested models:
  - Estimate model 1
  - Use `estimates store` *somename* to store the results
  - Estimate model 2
    - Storing the results is optional
  - Use `lrtest` *somename*
- Warning: Stata will *not* complain if the models are not nested

---

Nested Models, Example

- We'll reuse some old results, to illustrate how this works
- Let's restore the results from the full model

```
. estimates restore low_alsur

(results low_alsur are active now)
```

- We can see what they are easily:

```
. estimates
```

```
------------------------------------------------------------------------------
Model low_alsur
------------------------------------------------------------------------------

Logistic regression                               Number of obs     =        189
                                                  LR chi2(6)        =      23.07
                                                  Prob > chi2       =     0.0008
Log likelihood = -105.79896                       Pseudo R2         =     0.0983

------------------------------------------------------------------------------
         low |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         age |  -.0197546   .0344038    -0.57   0.566    -.0871848    .0476756
         lwt |  -.0113698   .0064025    -1.78   0.076    -.0239184    .0011788
       smoke |   1.022659   .3833948     2.67   0.008     .2712193    1.774099
          ui |   .7669792   .4402573     1.74   0.081    -.0959094    1.629868
    _Irace_2 |   1.256841   .5165319     2.43   0.015     .2444573    2.269225
    _Irace_3 |   .9294705   .4213411     2.21   0.027     .1036571    1.755284
       _cons |   .0104206   1.135112     0.01   0.993    -2.214357    2.235199
------------------------------------------------------------------------------
```

- We can see what the results from the other model were

```
. estimates replay low_sur

------------------------------------------------------------------------------
Model low_sur
------------------------------------------------------------------------------

Logistic regression                               Number of obs     =        189
                                                  LR chi2(4)        =      18.80
                                                  Prob > chi2       =     0.0009
Log likelihood = -107.93404                       Pseudo R2         =     0.0801

------------------------------------------------------------------------------
         low |      Coef.   Std. Err.      z    P>|z|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
       smoke |   1.080361   .3740433     2.89   0.004     .3472495    1.813472
          ui |   .8834084   .4329481     2.04   0.041     .0348457    1.731971
    _Irace_2 |   1.116041   .4907334     2.27   0.023     .1542216    2.077861
    _Irace_3 |   1.072471   .4069078     2.64   0.008     .2749467    1.869996
       _cons |  -1.964536   .3653468    -5.38   0.000    -2.680603   -1.248469
------------------------------------------------------------------------------
```

- Now we can run the likelihood ratio test

```
. lrtest low_sur

Likelihood-ratio test                             LR chi2(2)  =       4.27
(Assumption: low_sur nested in low_alsur)         Prob > chi2 =     0.1182
```

- Note: we could have run the liklihood test without restoring the first set of estimates—this was done here to show how restoring works

Non-nested Models

- Information criteria (AIC and BIC) can be used to compare models (not necessarily test them)
- This is also done with `lrtest`, but using the`stat` option

```
. lrtest low_sur, stat

Likelihood-ratio test                                 LR chi2(2)  =      4.27
(Assumption: low_sur nested in low_alsur)             Prob > chi2 =    0.1182

-----------------------------------------------------------------------------
       Model |    Obs    ll(null)   ll(model)     df          AIC         BIC
-------------+---------------------------------------------------------------
     low_sur |    189    -117.336    -107.934      5     225.8681    242.0768
   low_alsur |    189    -117.336    -105.799      7     225.5979    248.2902
-----------------------------------------------------------------------------
               Note:  N=Obs used in calculating BIC; see [R] BIC note
```

- Many models may be fit and named using `estimates store` and then later compared
- Some care must be taken if the models are fit with different commands

---

A Note on `estimates restore`

- `estimates restore` restores most estimation results well enough, but it does not restore `e(sample)`
  - Why not? One point of `estimates restore` is for using a model on out-of-sample results (such as a testing dataset)
- You need to use `estimates esample:` to restore `e(sample)` to its proper state
  - Don't forget the colon!
  - Also: use the `replace` option if you have estimates in memory already:

```
. estimates esample:, replace
```

There is More!

- Some Notes
  - What has been shown here works across nearly all estimation commands
    - Part of the strength of consistent command syntax
  - There are still more post-estimation commands which can be used across models. See `help postestimation commands`

11.4 Command-Specific Postestimation

11.4.1 Command-Specific Postestimation

Command-Specific Postestimation

- Each estimation command has some special, specific postestimation commands
- Only a couple will be touched on here

`logit` Postestimation

- There are two very worthwhile postestimation commands associated with `logit` (or `logistic`)

```
. xi: logit low age smoke ui i.race
```

- The `lroc` command will draw a ROC curve for assessing dicriminatory ability

```
. lroc

Logistic model for low

number of observations =       189
area under ROC curve   =    0.6631
```

- The `estat gof` for goodness of fit

```
. estat gof

Logistic model for low, goodness-of-fit test
--------------------------------------------

      number of observations =       189
 number of covariate patterns =        97
            Pearson chi2(91) =       102.14
                 Prob > chi2 =         0.1996
```

- `estat class` can be used for classification results:

```
. estat class

Logistic model for low

              -------- True --------
Classified |         D            ~D  |      Total
-----------+--------------------------+-----------
     +     |        17            16  |         33
     -     |        42           114  |        156
-----------+--------------------------+-----------
   Total   |        59           130  |        189

Classified + if predicted Pr(D) >= .5
True D defined as low != 0
```

```
------------------------------------------------
Sensitivity                     Pr( +| D)   28.81%
Specificity                     Pr( -|~D)   87.69%
Positive predictive value       Pr( D| +)   51.52%
Negative predictive value       Pr(~D| -)   73.08%
------------------------------------------------
False + rate for true ~D        Pr( +|~D)   12.31%
False - rate for true D         Pr( -| D)   71.19%
False + rate for classified +   Pr(~D| +)   48.48%
False - rate for classified -   Pr( D| -)   26.92%
------------------------------------------------
Correctly classified                        69.31%
------------------------------------------------
```

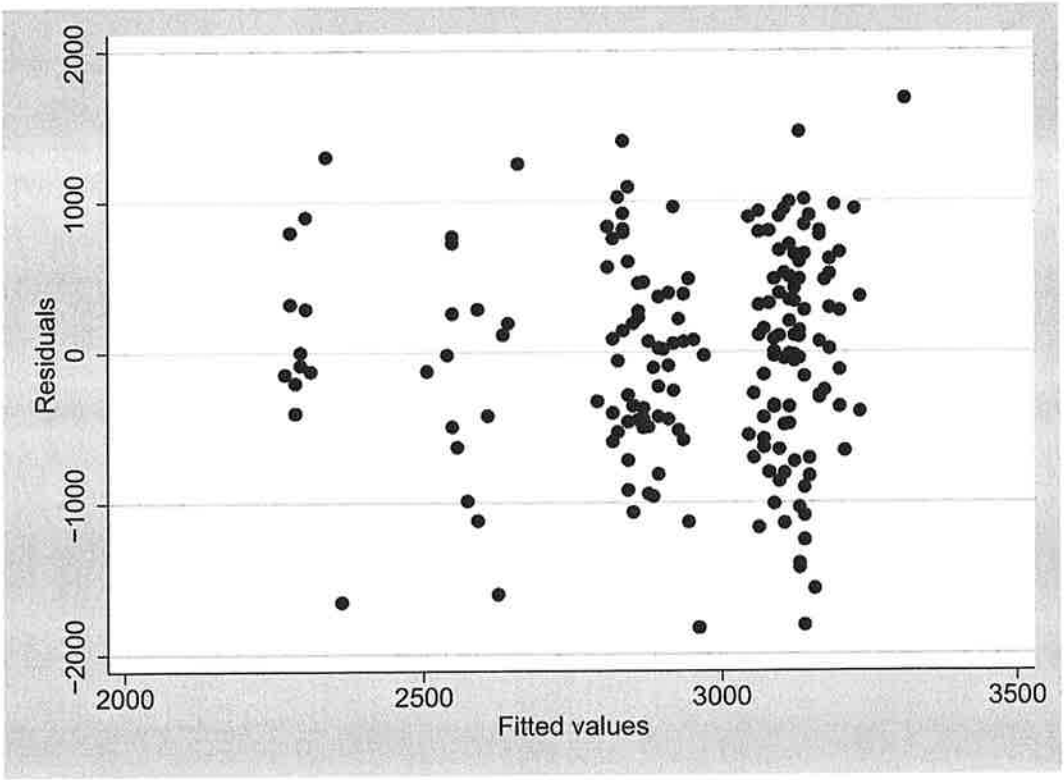
`regress` Postestimation

- There are multiple useful graphs which can be drawn after regression
- These mostly are graphs for visually checking model assumptions and influence statistics
- See `help regress postestimation`
- An example

```
. regress bwt age ui smoke
```

```
      Source |       SS       df       MS              Number of obs =     189
-------------+------------------------------           F(  3,   185) =    7.92
       Model |  11367926.2     3  3789308.73           Prob > F      =  0.0001
    Residual |  88547372.4   185  478634.445           R-squared     =  0.1138
-------------+------------------------------           Adj R-squared =  0.0994
       Total |  99915298.6   188  531464.354           Root MSE      =  691.83

------------------------------------------------------------------------------
         bwt |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
         age |   8.503649   9.557205     0.89   0.375    -10.35147    27.35877
          ui |  -548.5662    142.308    -3.85   0.000    -829.3215    -267.811
       smoke |  -253.7586   103.3842    -2.45   0.015    -457.7222   -49.79511
       _cons |   2927.301   235.0232    12.46   0.000     2463.631    3390.972
------------------------------------------------------------------------------
```

```
. rvfplot
```



11.5 Conclusion

11.5.1 Conclusion

Conclusion

- Estimation is a rich topic in Stata
- The estimation commands have a consistent syntax
- There is a common set of postestimation commands which can be used following estimation commands
- There are some command-specific postestimation commands

Finishing Up

- Close your log file

Lesson 12

Stata Graphics and the Graph Editor

12.1 Intro

12.1.1 Goals

Goals and Objectives

- Making simple univariate graphs.
- Making simple bivariate graphs.
- Using dialogs to build graphs for data exploration.
- Using the Graph Editor to mark up graphs.

12.1.2 Underlying Choices

Command or Dialog?

- Commands for quick graphs
- Dialogs for complex or specially formed graphs

Dialogs or Editor?

- Dialogs for reproducible results of general graph layout
- Editor for specific graphs based on specific data and quick markup

---

12.1.3 Startup

Keep good records

- Start a log file.
  - The log file will hold the commands but not the graphs!

Datasets

- Start with the Hosmer and Lemeshow birthweight dataset

```
webuse lbw.
```

- This dataset is poorly labeled.
  - Define a value label `smoker` to associate 0 with "non-smoker" and 1 with "smoker", and then attach it to the `smoke` variable.

12.2 Simple Graphs

12.2.1 Simple Univariate Graphs, Saving, and Exporting

Histograms via Command window

- Would like a histogram of `bwt`.
- Simple command: `histogram` *varlist*.
  - Here: `histogram bwt` or `hist bwt` for the impatient.
- Simple enough for a quick glance.
- Data look roughly normal.

Histograms via Dialogs

- **Graphics > Histogram**
  - Since we know the name of the command `db histogram` will also work.
  - In general, `db` *commandname* will yield the proper dialog box.
- There is a lot to choose from, here!

Adding a Normal Overlay

- Try clicking the ***Density plots*** tab, and selecting *Add normal density plot*.
- Click the **Submit** button to draw the graph but keep the Graph window.
- A normal curve with the same mean and standard deviation as the data is drawn over the histogram.
- The `normal` option could be used from the Command window.

Adding a Kernel Density

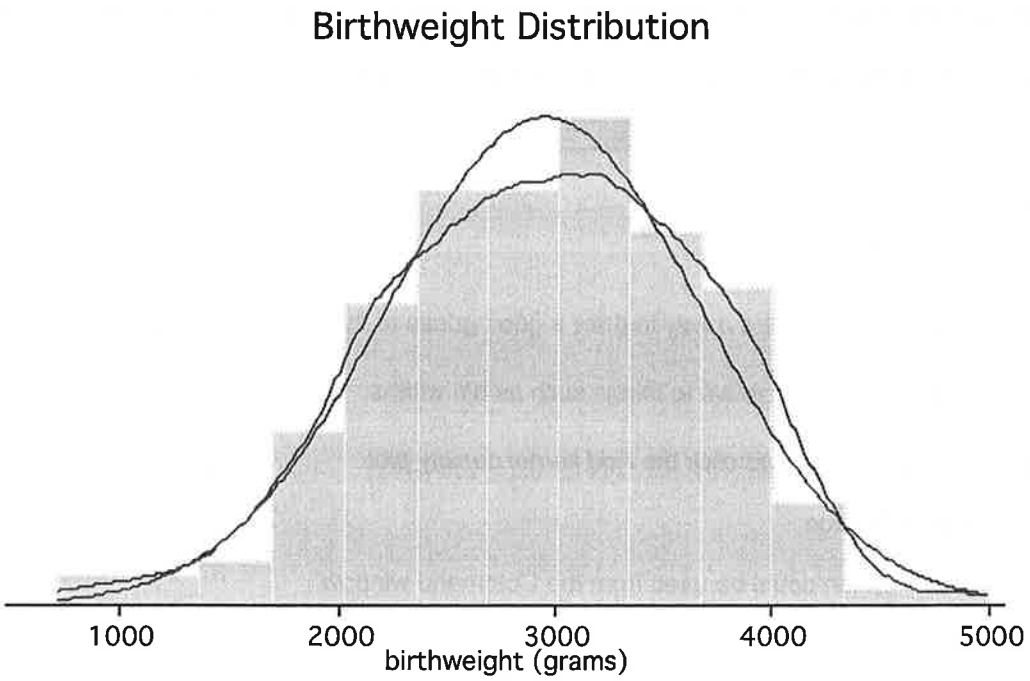
- Kernel density estimators are a way to draw a good guess at the underlying density.
  - Advantage: not as sensitive to things such as bin widths.
- Bring the dialog forward, and click the *Add kernel density plot*.
- Click the **Submit** button.
- The `kdensity` option could be used from the Command window.

Completing the Graph

- For the graph to be complete, it needs a good title and it needs to be well-labeled.
- For a histogram where shape is the important feature, the y-axis is irrelevant.
  - Click the ***Y axis*** tab and select *Hide axis*.
- We would like a title at the top which is visible from afar.
  - Click the ***Titles*** tab.
  - Type `Birthweight Distribution` in the *Title* field.
- Would like to quote the source.
  - Type `Source: Hosmer and Lemeshow` in the *Caption* field.
- Click the **Submit** button.

The Well-Labeled Graph

- The fruit of our labor:



From: Hosmer and Lemeshow

12.2.2 Saving, Exporting and Modifying Graphs

Changing the Look

- Stata's graphs are governed (in general) by *schemes*.
- These can be found under the ***Overall*** tab.
- Example: try a scheme to see the change.
- Schemes are useful, but rather a pain to make on your own.
- We'll see that using the Graph Editor can be used for some changes.

Saving Our Work

- To save the graph as a Stata graph type `graph save "`*filename*`"` in the Command window.
  - Use the `replace` option to overwrite an existing file.
  - Right-clicking on the graph will bring up a contextual menu which also allows saving the graph.
- This saves the graph as a live graph which allows editing at a later date.
  - Add the `asis` option to freeze the graph.

General Idea

- To put the graph in a document, it must be translated into a useful format.
- This can be done using the mouse or by using the Command window
  - No surprise there!

Exporting a Graph with the Mouse

- The simple way to do the mouse is to right-click on a graph and select **Save Graph**...
- Choose the desired format
- For Windows, this format is the Enhanced Windows Metafile (emf) format.
- For a Mac, this format is Portable Document Format (pdf) format.
  - Same as above, but use `.pdf` as the extension.
- For Unix, this format is usually Encasulated Postscript (eps) format.
  - Same as above, but use `.eps` as the extension.
- Try this now.

Exporting a Graph from the Command window

- The command is `graph export "`*filename*`"` on all platforms
- The choice of format is made by choosing the file extension
  - For MS Window: `.emf`
  - For the Mac: `.pdf`
  - For Unix: `.eps`

Modifying Graphs' Fonts

- Many times, you might want to change the font on the exported graph
- If you want to change it in general, change it from the **Graph Preferences**
  - To be sure that the fonts match on screen, when copied and when printed, be sure to change the font on all three tabs.
- If you want to change it for a particular graph, the `fontface` option will allow you to specify a font to use for all formats other than PDF.

Working with Multiple Graphs

- Stata typically has just one graph open at a time.
- We can have more than one open if we *name* the graph.
- Give the graph a name in the ***Overall*** tab.
- Click the **Submit** button.
- This makes a new, named graph window that will not be overwritten by a new graph.
- As we'll see, this is useful for combining multiple graphs.

Preparing for Another Graph

- Delete the name the graph from the ***Overall*** tab.
- This will make subsequent graphs appear in the usual Graph window.

Comparing Subpopulations

- Click the ***By*** tab.
- Check the *Draw subgraphs for unique values of variables* checkbox.
- Choose the `smoke` variable.
- Click the **Submit** button.
- The graph is OK, but hard to use for comparisons.
  - It would look better if the graphs were above one another.

Changing the Subgraph Layout

- Click the **Subgraph organization** button.
- Change the *Rows/Columns* to *Columns*
- Click the **Accept** button.
- Click the **Submit** button.
- Rather anti-climactic, no?

Some Fixes

- Getting rid of the legend
  - Click the ***Legend*** tab.
  - Select *Hide legend*
- Getting rid of the y-axes
  - Click the **Subgraph axes** button.
  - In the ***Show axes*** group, change the *Y axes* to *No*
    - This doesn't completely work, though it ought to: so shut off all other Y axis info
  - Click the **Accept** button.
- Making the taller
  - Select the ***Overall*** tab
  - Change the width to 5 inches and the height to 8 inches.
  - Click the **Submit** button.
- Save this if you like.

12.2.3 Back to Univariate Graphs

The Boxplot

- Select **Graphics > Box plot.**
- Select the `bwt` variable.
- Click the ***Categories*** tab.
- Check the *Group 1* checkbox.
- Select the `smoke` variable.
- Check the *Group 2* checkbox.
- Select the `race` variable.
- Click the **Submit** button.

The Boxplot (cont.)

- As you can see, the command is not too hard, here.
- You could alter this in a similar fashion to the histogram.
- We'll keep moving on.

12.2.4 Simple Bivariate Plots

Starting in the Command Window

- The `scatter` *yvar xvar* command will make a simple scatterplot.
- Try this `scatter bwt lwt`.
- Not bad for checking out a quick scatter.

Scatterplots and More

- Let's make a scatterplot of the birthweight against the weight of the mother.
- We'll see that we'll be able to add regression lines and such directly to the plot.
- We'll see that we can overlay plots easily.
- We'll see that we can change points and use this to make some nice graphs.

The *twoway* Dialog

- Select **Graphics > Twoway graph (Scatter, Line, etc.)**
- This is the main place to make twoway graphs, regardless of the type.
- The dialog is new in Stata 10—in Stata 9, the dialogs are a bit less clear.

Making a First Plot

- Click the **Create...** button.
  - Yow! All the possible twoway graphs are collected here in one spot.
- Be sure the *Basic plots* and the *Scatter* items are chosen.
- Select `bwt` as the *Y variable* and `lwt` as the *X variable*
- Click the **Accept** button.
- Click the **Submit** button.

Noting the Command

- The command looks slightly different from what we had typed.
- `twoway` really is the command.
- The parentheses are to get ready for having multiple overlaid graphs.

Making an Overlaid Graph

- Let's overlay a regression line.
- We need to create a second plot which will be overlaid.
- Click the **Create...** button, again.
- Click the *Fit plots* radio button.
- Enter the same X and Y variables as before.
- Click the **Submit** button.
- If things worked fine, click the **Accept** button.
- Name this graph as bwtlwt (save if you like).
- Take a look at the final command.

Disabling Overlays

- Select the plot.
- Click the **Disable** button.
- Click the **Submit** button.

The `by` Option Revisited

- Click the ***By*** tab.
- Check the *Draw subgraphs for unique values of variables* checkbox.
- Click the **Submit** button.
- See the interaction of smoking and maternal weight.

Playing with Colors and Plot Symbols

- Select Plot 1
- Click the **Edit** button.
- Click the **Marker properties** button.
- Play around for a while with the items in the ***Marker properties*** group of the dialog box.
  - Don't fool with adding labels to markers, yet.
- Some symbols may look a bit funny on the screen, but they print and export just fine.

Looking at a Third Variable

- Check the **Add labels to markers** checkbox.
- Select the variable `smoke` as the label.
- Click the **Submit** button.
- A bit busy...

Fixes

- We can remove the value labels (temporarily) from the `smoke` variable.
  - Type the command `label values smoke` to remove the value label from the variable.
  - The value label is still defined, and we can reattach it later.
- We can replace the markers by the values.
  - Move the markers over their true valuees
    - Change the *Label position* to *Middle*
    - Click Submit to see the result.
  - Turn off the default markers
    - Select the symbol *None*
- Click the **Submit** button.
- Pretty Slick!

Places to Learn More

- Stata has a very nice set of examples here:

  `http://www.stata.com/support/faqs/graphics/gph/statagraphs.html`

- UCLA has something similar, but slightly older:

  `http://www.ats.ucla.edu/stat/stata/library/GraphExamples/default.htm`

Other Methods

- Time permitting, we'll try using different colors to show who the smokers are.
- Hint: it involves overlaid graphs with `if` qualifiers!

12.2.5 Using the Graph Editor

Stata 10 Has a Graph Editor!

- This is brand new—it does not exist in Stata 9.
- We'll experiment with the `bwtlwt` graph from earlier.
- Most of this will be point and click, and hard to describe, so the notes are sparse!

An Important Tip

- The Graph Editor is *modal*, which means once started, you cannot enter any commands from the Command window.
- If you want to view help files while using the editor, you must open a viewer, then get help!

Start the Graph Editor

- Click the Start Graph Editor button.
- Note the tools on the left-hand side.
- Note the objects on the right-hand side.

Selecting an Object

- Select the fitted line.
- It becomes outlined.
- It's object name becomes highlighted.
- A contextual toolbar appears at the top of the graph.
  - This contains properties which can be edited for this object.
- Change the color to *Cranberry*

Add another line

- Select the line tool
- Draw the line
- To change the new line's properties, select it first.
- Experiment!

The Graph Editor and Commands

- The Graph Editor does not save a command to be used later.
- There is good reason for this: there are edits which can be done in the graph editor which are not easily done from the Command window.

The Record Button

- There is a record button, however, which can be used to record edits.
- Saved recordings can be applied to other graphs by appending the `play()` option.
  - Note that all drawn objects have fixed locations based in the units of the graph.
  - This is useful for changing overall appearances, though
- By default, the recordings are saved in the `grec` folder in your `PERSONAL` folder.
  - So... if you want to give them to a friend, this is where they will be.

Recordings as Cheap Schemes

- Recordings can be used as schemes, especially if there is a single type of graph you make often.
- Be forewarned, however, that a recording for a twoway graph will most likely not work for, say, a histogram.
  - This means one recording would be needed for each type of graph!

12.3 Conclusion

12.3.1 Underlying Choices (again)

Command or Dialog?

- Commands for quick graphs
- Dialogs for complex or specially formed graphs

Dialogs or Editor?

- Dialogs for reproducible results of general graph layout
- Editor for specific graphs based on specific data and quick markup
- Recorder for a quick way to make a scheme for a particular type of plot

---

12.3.2 Places to Learn More

12.3.3 Questions?

Stata Web site, FAQs, Graphs

Lesson 13

Subscripting and by

13.1 Introduction

13.1.1 Goals

Goals

- Learning about working across observations
- Learning subscripting
- Learning more about the `by` prefix command

13.1.2 Getting Started

Getting Started

- Create a folder called `subscripting`
- Start a log, also called `subscripting`
- Download the 2 small datasets from the `bysubscripting` package.

13.2 Subscripting

13.2.1 Background

Using Subscripting When Generating - Purpose

- Suppose the values of a variable could be predicted (or estimated) by earlier values.

  - For evenly spaced time increments, these would be called autoregressive models.
  - For longitudinal data, these are called transition models.
  - This means that lagged variables are needed.

- What is needed?

  - Well, to look at the previous observation, use something like

    `generate` *lagged = original at the last observation*

Another Use

- Suppose that the interest was only in a relative change from observation to observation.
- If the times were evenly spaced, we would need something like

```
generate delta = value_ this_time/value_last_time
```

- **Needed:** a way to generate across observations.

How to Start?

- In order to figure out how to work across observations, we need to know:

  - How Stata words normally
  - How to refer to another observation.

The Usual Way Stata `generates`

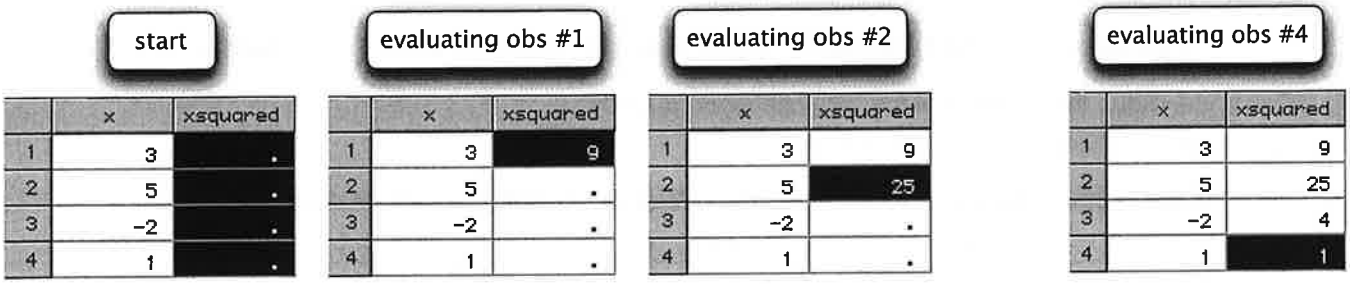
- Example: `gen xsquared = x^2`

  - What happens?

    * Stata creates a new variable `xsquared`, and makes it entirely missing.
    * Stata goes through each of the observations in the dataset, and at each observation, it looks at the value of x, squares it, and places it in the variable xsquared.

Pictures

- Here are pictures for a small dataset:



	x	xsquared
1	3	.
2	5	.
3	-2	.
4	1	.

	x	xsquared
1	3	9
2	5	.
3	-2	.
4	1	.

	x	xsquared
1	3	9
2	5	25
3	-2	.
4	1	.

	x	xsquared
1	3	9
2	5	25
3	-2	4
4	1	1

Subscripting to the Rescue

- Something like this cannot work for lagging variables, since using a simple `generate` command works within each observation.
- Need a method for explicitly referring to other observations.
- Need to use the observation numbers—i.e. subscripts

Rephrasing the Usual `generate`

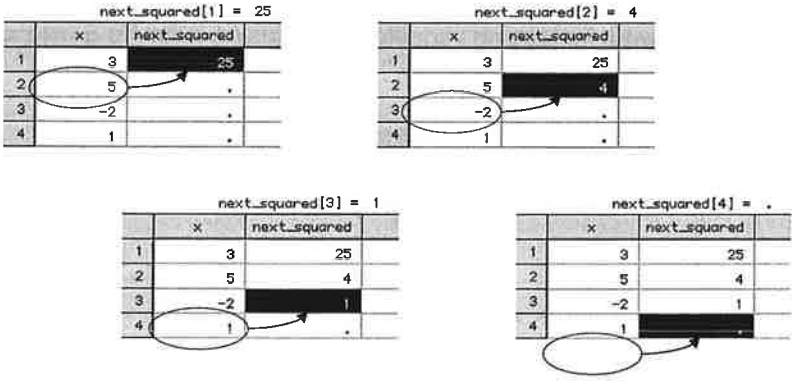
- Example: `gen xsquared = x^2`
  - Stata creates a new variable xsquared, and makes it entirely missing.
  - Stata goes through each of the observations in the dataset, and at each observation, it looks at the value of x, squares it, and places it in the variable xsquared.
    - Stata puts x[1]^2 into xsquared[1]
    - Stata puts x[2]^2 into xsquared[2]
    - etc.
- These is all good and well, but how can we refer to other observations in a *relative* fashion?

Some Special Friends

- Here are some so-called system variables which help:
  - `_n` is the number of the current observation
  - `_N` is the total number of observations
- In expressions, variables without any explicit subscript have the implicit `[_n]` subscript.
  - So... the example of the usual generate is really
  - `gen xsquared = x[_n]^2`
- To get to other observations, use a relative reference using the `_n` system variable!

Continuing the Example

- What if you wanted the squared value of the next observation?
- `gen next_squared = x[_n+1]^2`



Example: Times Between Visits

- Setup: We have a `visitdt` variable containing dates of visits—and we want the time between visits.
- We'll make a dataset on the fly

```
. clear
. set obs 8

obs was 0, now 8

. set seed 948924
. gen visitdt = 17500 + int(runiform()*150)
. format visitdt %td
```

Using the Subscripting

- For the time between visits, we need to subtract the previous visit date from the current visit date:

```
. sort visitdt
. gen datediff = visitdt - visitdt[_n-1]

(1 missing value generated)
```

- Take a look to be sure this worked:

```
. list
```

```
     +----------------------+
     |   visitdt   datediff |
     |----------------------|
  1. | 05dec2007          . |
  2. | 02feb2008         59 |
  3. | 12feb2008         10 |
  4. | 10mar2008         27 |
  5. | 18mar2008          8 |
     |----------------------|
  6. | 23mar2008          5 |
  7. | 28mar2008          5 |
  8. | 30mar2008          2 |
     +----------------------+
```

13.2.2 The by Prefix Command

Extending the Problem

- Now suppose that there were multiple patients, and we needed the time between visits within each patient.
- Open up the `manyvisits` dataset

```
. clear
. use manyvisits
```

- We can see the visits for each of the patients by using `list` with the `sepby` option

```
. list, sepby(id)
```

```
     +----------------+
     | id     visitdt |
     |----------------|
  1. | 10   03feb2001 |
  2. | 10   13feb2001 |
     |----------------|
  3. | 22   23oct2000 |
     |----------------|
  4. | 35   10jan2001 |
  5. | 35   24jan2001 |
  6. | 35   31jan2001 |
     +----------------+
```

If Only

- There were one minidataset for each person, we could use the command from before

  ```
  gen time_between = visitdt-visitdt[_n-1]
  ```

  on each of these minidatasets to get:

	id	visitdt	time_between
1	10	03feb2001	.
2	10	13feb2001	10

	id	visitdt	time_between
1	22	23oct2000	.

	id	visitdt	time_between
1	35	10jan2001	.
2	35	24jan2001	14
3	35	31jan2001	7

Stata's `by` Prefix Command

- We need some way of running the same command on subsets of the main dataset.
  - The subsets can be defined by one or more variables.
- The `by` prefix command is a prefix clause, because it comes before Stata command.
- Look at `help by`
- `by` *varlist1* [(*varlist2*)][, `sort`]: *stata_command*

The Result of Using a `by` prefix command

- If the `sort` option is specified, the data are sorted by the combined varlist of *varlist1 varlist2*
  - Otherwise it checks to see if the data are sorted by these variable lists, and issues an error if the data are not sorted.
- It then behaves as though the dataset were split into the groups defined by the different values of *varlist1*
- It then runs the stata_command on each subset of the dataset.

Solution!

- Using the following command on the original dataset will do the trick:

```
. by id (visitdt), sort: gen time_between = visitdt-visitdt[_n-1]

(3 missing values generated)
```

  - Why are there 3 missing values?

- Looking at the results:

```
. list, sepby(id)
```

```
     +---------------------------+
     | id     visitdt   time_b~n |
     |---------------------------|
  1. | 10   03feb2001          . |
  2. | 10   13feb2001         10 |
     |---------------------------|
  3. | 22   23oct2000          . |
     |---------------------------|
  4. | 35   10jan2001          . |
  5. | 35   24jan2001         14 |
  6. | 35   31jan2001          7 |
     +---------------------------+
```

More Uses of `_n` and `_N`

- The `_n` and `_N` variables come in handy in other situations, among other things,
  - Marking the first or last observation in a group
  - Looking for duplicates
- Some of the uses are in built-in Stata commands—the demonstration here is to open an otherwise invisible window

---

`_n` and `_N` and a `by` prefix command

- Try this:

```
. by id (visitdt), sort: gen nid = _n
. by id (visitdt), sort: gen Nid = _N
```

- Take a look at what is now here:

```
. list, sepby(id)
```

```
     +-------------------------------------+
     | id     visitdt   time_b~n   nid   Nid |
     |-------------------------------------|
  1. | 10   03feb2001          .     1     2 |
  2. | 10   13feb2001         10     2     2 |
     |-------------------------------------|
  3. | 22   23oct2000          .     1     1 |
     |-------------------------------------|
  4. | 35   10jan2001          .     1     3 |
  5. | 35   24jan2001         14     2     3 |
  6. | 35   31jan2001          7     3     3 |
     +-------------------------------------+
```

Marking the First or Last Observation

- Studying the above gives the following tools:
- Marking the first observation in a group:

```
. by id (visitdt), sort: gen first = _n == 1
```

- Marking the last observtation in a group:

```
. by id (visitdt), sort: gen last = _n == _N
```

Checking for Multiple Visits

- If we wanted to mark the visits of those who had multiple visits, this would be simple enough:

```
. by id, sort: gen multi = _N-1
```

- `_N-1` was used so that single visit patients would have their only visit marked with a 0—making listing of the "interesting" patients simpler:

```
. list if multi, sepby(id)
```

```
     +-------------------------------------------------------------+
     | id     visitdt   time_b~n   nid   Nid   first   last   multi |
     |-------------------------------------------------------------|
  1. | 10   03feb2001          .     1     2       1      0       1 |
  2. | 10   13feb2001         10     2     2       0      1       1 |
     |-------------------------------------------------------------|
  4. | 35   10jan2001          .     1     3       1      0       2 |
  5. | 35   24jan2001         14     2     3       0      0       2 |
  6. | 35   31jan2001          7     3     3       0      1       2 |
     +-------------------------------------------------------------+
```

- This same logic can be used for checking the uniqueness of a primary key in a file
  - Substitute the key variables for `id` above

13.3 Conclusion

13.3.1 Conclusion

Notes

- Subscripting and `by` are very handy, but not very obvious
- If you find yourself tempted to try and work through a Stata dataset observation by observation by using a loop, you probably can get something much faster with `by`

Finishing up

- Close your log file

Lesson 14

Basic Do-files

14.1 Introduction

14.1.1 Goals

Goals

- Learning strategies for using do-files
- Learning to work using do-files

14.1.2 Getting Started

Getting Started

- Make a folder called `dofiles`
- Start a log with the same name

14.2 Automation

14.2.1 Automation of Tasks

Stata has two possibilities for automating tasks:

- **do-files** are for batch processing.
  - At their simplest, do-files are just a series of Stata commands which are saved in a text file with an extension of .do.

- **ado-files** are for writing new Stata commands.
  - Stata has tools for making ado-files fit Stata's syntax.

What Here?

- We will concentrate on do-files.
- We will learn to work from the Do-file Editor.

14.2.2 Do-files

Something New

- We will edit our do-files in the Do-file Editor.
- You can open a Do-file Editor window by clicking the Do-file Editor button, or by typing the `doedit` command.
- Do this now.

A Simple Do-file

- Type the following into the Do-file Editor:

```
sysuse auto
regress mpg weight
display "I am now done"
```

- Save the resulting file as the do-file `simple.do` in your current working directory.

Running the Do-file from the Command Window

- Run the do-file by typing

```
do simple
```

- Stata echos each command to the Results window and then executes it.
- This is what you need for record-keeping.

A Useful Tip

- It is worth putting an `exit` command at the bottom of your do file. There are two benefits:
  - Stata needs a final EOL to run the final line. Putting in a (silent) exit command guarantees the last useful command runs.
  - You can use the space below the `exit` command as a place to store snippets, because the do-file will end when it reaches `exit`

Running the Do-file from the Do-file Editor

- Clicking the **Do** Button will run the file as it stands.
- You can select commands in the Do-file Editor, and do just those commands.
- You can run the commands silently by using the **Run** button.

Aha! A Different Way to Work

- It should now be possible to work from the Do-file Editor instead of the Command window.
- Add Commands as you think they will be useful, and then rerun the code from the Do-file Editor.

A Problem: `--more--`

- Add a `codebook` command before the `regress` command in your do-file
- Run the do-file.
- You will likely get a `more` condition.
  - If you do not, you might have a space or some other text in the Command window... clear it out and try again

---

A Solution: `set more off`

- Add the line `set more off` to the top of your do-file.
- This makes sure the do-file runs to its completion.
  - Or at least to its first error!
- When you are sure it works, get rid of the `codebook` command

A Problem: Data

- Change the do-file to generate a new variable which is 1/mpg, and then regress this new variable against weight.
- Try running the do-file again.
- Failure! Stata will not reload the `auto` dataset.

A Solution

- A *temporary* solution is to add a `clear` command to the top of your do-file.
- This is **dangerous**, because it clears out data, even if you don't want to!
  - Leave it in as long as you are putting together the do-file, then take it out.

14.2.3 Using Do-files for Reproducibility

Logging

- For reproducibility's sake, you should keep a log of each do-file that runs.
- Add the line

  `log using simple, name(simple) replace`

  below the `set more off` line of your `simple.do` file.
  - This names your log file, and keeps the new log file from interfering with the non-named, special log file you opened at the start of your session.
  - The `replace` option ensures that you will write a new simple log whenever you run simple.do.
- Add the line `log close simple` to the bottom of your do file.

How Does It Look?

- By now, your do file should close to this

```
. type simple.do
set more off
log using simple, name(simple) replace
clear
sysuse auto
gen gpm = 1/mpg
regress gpm weight
display "I am now done"
log close simple
exit
```

A Problem with Errors

- Put a garbage command in your do-file.
- Run the do-file.
- Run the do-file again.
- Uh-oh! The log file is still open!

Trying to Fix the Problem

- We need a method of guaranteeing that the log-file is closed as we start.
- To do this, we would need to have a `log close...` command at the top of the file.
- A conundrum: if the log file does close, this will now cause an error.
- What to do?

Keeping Errors from Ruining Your Day

- Putting `capture` before a command captures errors (and all output) from the command.
- This is our key: add this just before your log command:

```
capture log close simple
```

- Now run your do-file.

Looking at your log

- You can now look at the log from your do-file:

```
view simple.smcl
```

What If You Upgrade Stata?

- For true reproducibility (and a lack of headaches) we'd like this to keep working under Stata 11, Stata 12, etc.
- One of Stata's best features: version control
- Put a `version 10.1` command at the top of the do file, and it will always work and always work like it does today.
  - Editorial remark: Stata's version control is really truly amazing!

Skeleton for Reproducibility

- You now have a skelton for reproducibility and logging!

```
* replace simple with the name of your do-file
version 10.1
set more off
capture log close simple
log using simple, name(simple) replace
clear /* should be temporary */
----- put your code here -----
log close simple
exit
```

Nesting Subprojects

- Do files may "do" other do files.
- If all of these are built **with named logs**, then each log will contain the output from all its children.
- This means you can have a parent do-file for a project which then does many sub-project do-files (which then does, etc.).
  - This allows working on small, local pieces after running a partial do-file.

14.2.4 Subroutines

Subroutines in Do-Files

- You can put subroutines in your do file by putting in a block of code for each subroutine:

  `capture program drop` *subroutine*
  `program define` *subroutine*
  —code goes here—
  `end`

- You can call these subroutines by name within your do-file.
- You should put the subroutines at the top of your do-file just after you open the log (so their definitions are captured).

Silly Example

- Put a subroutine called `howdy` in your simple.do file which has one line of code:

  ```
  display as result "Howdy Doody!"
  ```

- Put one or two lines in your simple.do file which simply are `howdy`

14.2.5 Notes

Comments and Continuation

- Can use `/* */` to enclose comments.
  - These can be multiline commends and may be nested.
- Can use `//` at the end of a line as a comment.
  - The comment lasts from the `//` until the end of the line.
- Can use `///` for line continuations.
  - Can put commented text after the `///`, also, though this is no very common.

Style Suggestions

- Many people put their own version numbers and time-stamps at the top of their do-files..
  - These start with `*!`
    - This has a special meaning when writing ado-files—hence it's use here.

What More?

- We cannot do much of value until we learn about macros, looping and branching.

14.3 Conclusion

14.3.1 Conclusion

Conclusion

- You can get the skeleton do-file from the course site:
- In the results, click the `basicdofile` link and download the skeleton

---

Finishing Up

- Close your log file

Lesson 15

Macros and Looping

15.1 Introduction

15.1.1 Goals

Goals

- Learning about some common programming tools
  - Containers
  - Branching and Looping
- Seeing how Stata implements these

15.1.2 Startup

The Usual

- Make a new folder called `macros`
- Start a log
- Download the `macros` package
- Open up the auto dataset

15.2 Stata's Containers

15.2.1 Container Concepts

Why Do We Need Containers?

- We'll eventually need to write more complex do-files and Stata programs
- So far, all we know how to manipulate are datasets
- We need something to be able to hold information which is needed for the program, but which is not part of the dataset
- Such "floating objects" have many different names—we'll call them *containers*

How Does Stata Implement Containers?

- Stata has a few types of containers, depending on what needs to be stored
- *Macros* are simple text (string) containers
  - There are global and local macros. We'll talk only about the latter.
- *Scalars* are for holding single numbers
  - Technically, they can hold strings, too, but this is less than useful
- *Matrices* are for holding multple numbers in one container

---

15.2.2 Macros

Local Macros

- We'll start with local macros
- A local macros are *local* in two ways:
  - Its definition lasts for the length of their containing context (do-file or ado-file)
  - Its definition is unknown in any other context
- Local macros can be used in the Command window

Global Macros

- Stata has global macros, whose definitions are known everywhere
  - We'll see some soon
- They are dangerous, because of their broad context
  - Could easily overwrite a global defined elsewhere
- We'll ignore these, for the most part

Definition of a Local Macro

- The most basic way to define a local macro is

  `local` *macroname* `"`*something*`"`
  - The " marks are optional, but recommended
- Macros can be quite long, over 1MB.
  - To see the exact length on your machine either
    - Type `creturn list` an look for `c(max_macrolen)`, or
    - Type `disp c(max_macrolen)`

Use of a Local Macro

- A macro's text is substituted via

  `` `macroname' ``
- Note the left and right quotes!
  - The left quote is above the **tab** key on US keyboards.
- Common terms for getting at the contents of a macro: *evaluating* and *dereferencing*

Try One Out

- Define a macro, and then `display` it (use your own name!)

```
. local name "Bill"
. display "Hello, my name is `name'"

Hello, my name is Bill
```

- This is simple enough: the local macro `name` contains your name—when dereferenced, it shows its contents

---

Dereferencing == Substitution Only

- Try this

```
. local sum "2+2"
. display "`sum'"

2+2
```

- Note that the macro really is nothing but a container—there is no magic evaluation.
  - Try leaving the quote off—the macro appears to be evaluated. Why?

Assigning vs. Storing

- Macros can be defined using an assignment clause:

  `local` *macroname* = *expression*

- This is sometimes needed for holding results of functions.
- Try this

```
. local whereat = strpos("this","is")
. display "`whereat'"

3
```

- Warning: When using an assignment clause, **the macro is limited in length to 244 characters**. Yow!
- Thus: avoid assignment when simple storage will do.

Deleting Macros

- The simplest way to destroy a macro is via

  `local` *macroname*

15.2.3 Macro Notes

Seeing What Is Defined

- To see what macros exist:

```
macro list
```

  - You might notice that there are some function keys assigned! We'll play with these in a bit

- You'll see that all local macros "official" names start with an underscore

---

Gotcha: Quotes Inside Strings

- If a double quote shows up in a string, how does the string get displayed?
  - With some extra formatting.
- Strings with quotes can be displayed by enclosing the outer quotes in macro expansion characters

```
. display `"something "with" quotes"'

something "with" quotes
```

- For this reason, if you don't know the content of macro, quote it doubly: `"`*macroname*'"'

Extended Macro Functions

- There are many extended macro functions—too many to go through.
- Look at `help macro`, and then click any `extended_fcn` link.
- We'll look through some until you understand the syntax.

Some Examples, Part 1

- Open up the auto dataset

```
. sysuse auto
```

- Copy a variable label into a local macro:

```
. local replabel : var label rep78
. display `"`replabel'"'

Repair Record 1978
```

- Put the names of the do-files in the working directory into a macro:

```
. local dofiles : dir . files "*.do"
. display `"`dofiles'"'

"child.do" "foreach_list.do" "foreach_var.do" "forval_gph.do" "forval_gph_compl
> ex.do" "forval_simple.do" "if_ex.do" "parent.do" "whaa.do" "while_ex.do"
```

Counting Tokens

- We can see count up numbers of words:

```
. local phrase "this that and the other"
. local howmany : word count `phrase'
. display "`howmany'"

5
```

  - No special quoting, because we know `howmany` is a number

Working with Sets

- We can work with sets

```
. local losethese "this and that"
. local newphrase : list phrase - losethese
. display `"`newphrase'"'

the other
```

Substituting Text

- We can substitute into strings

```
. local hmm : subinstr local phrase "th" "xx", all
. display `"`hmm'"'

xxis xxat and xxe oxxer
```

Length of Assigned Macros

- This was just mentioned, but it bears mentioning again: if a macro gets its value from an assignment or a usual Stata function, it's length is limited to 244 characters.
- Use extended macro functions when manipulating macro contents—these do not limit the length of the macros

---

Aside: Function Keys

- Function keys can be given single Stata commands via a global macro:

  `global F#` *Stata command* [`;`]

  - Putting the `;` at the end of the line acts like pressing **Enter**
- Example `global F5 macro list;`
  - Now press **F5**

15.2.4 Scalars

Defining a Scalar

- Scalars must be assigned:

  `scalar` *somename* =*exp*

- Try it:

```
. scalar anumber = 14
```

- If the assignment is missing, you'll get an odd error message.

Using a Scalar

- Scalar names are used with no special quoting
- Displaying alone

```
. display anumber^2

196
```

- Displaying in the midst of a sentence

```
. display "There are " anumber " days in 2 weeks"

There are 14 days in 2 weeks
```

Scalar or Variable?

- When Stata sees an unquoted name, it first tries to turn it into a variable name—if it cannot find a variable, it then looks to see if it can find a scalar
- This can cause some confusion if variable name expansion is turned on
  - `help set varabbrev`
- One convention: Name scalars either with Capitalized Names or with names that are in ALLCAPS.

Why Scalars?

- Scalars do a better job than macros at holding numbers
- Macros store printed version of the number
  - They are string containers, after all
- Scalars hold the actual number itself in binary
- This is more efficient and more accurate when the number must be reused
- This is worthless if all that is needed is a text container to hold the number
  - We'll see this when looping

15.2.5 Matrices

Defining a Matrix

- A matrix is defined via

  `matrix` *matrix_name* = *matrix_expression*

- The *matrix_expression* can be
  - Computations using other matrices
    - This happens after estimation, for instance
  - Entering values via

    `(#, ..., # \ #, ..., # \ ...)`

Displaying a Matrix

- Matrices can be displayed using

  `matrix list` *matrix_name*

- Matrices cannot be displayed in the middle of a `display` command
- Try this:

```
. regress mpg headroom trunk weight
```

```
      Source |       SS       df       MS              Number of obs =      74
-------------+------------------------------           F(  3,    70) =   44.16
       Model |  1598.77026     3  532.923421            Prob > F      =  0.0000
    Residual |  844.689197    70  12.0669885            R-squared     =  0.6543
-------------+------------------------------           Adj R-squared =  0.6395
       Total |  2443.45946    73  33.4720474            Root MSE      =  3.4738

------------------------------------------------------------------------------
         mpg |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
    headroom |   .0049958   .6427576     0.01   0.994    -1.276944    1.286935
       trunk |  -.0968366   .1503073    -0.64   0.522     -.396615    .2029419
      weight |  -.0056531   .0007083    -7.98   0.000    -.0070657   -.0042404
       _cons |   39.68369   1.805258    21.98   0.000     36.08321    43.28416
------------------------------------------------------------------------------

. matrix list e(V)

symmetric e(V)[4,4]
            headroom       trunk      weight       _cons
headroom   .41313728
   trunk  -.05024102   .02259229
  weight  -.00003154  -.00005714   5.017e-07
   _cons  -.45022098   .0121268  -.00063425   3.2589579
```

Niceness of Matrices

- Stata's matrices have row and column names
- These follow matrices through computations
- Can be used for building matrices and displaying results

Computing with Matrices

- Stata has a large number of commands made to work with matrices—see `help matrix`
- Stata's matrices have some limits—see `help matsize`
- If you want to dive into some powerful computing, Stata has its own matrix language—see `help mata`

15.3 Looping and Branching

15.3.1 Loops

`foreach` and `forvalues`

- The two main ways to loop are using `foreach` and `forvalues`
- `foreach` loops through words of macro.
- `forvalues` loops through values of a *numlist*

Structure of `foreach`

- `foreach` loops look generally like

```
foreach local macro name ... {
    do something with local macro
}
```

- It is best to go to `help foreach` to see all the ways that a foreach loop can be constructed

---

Working a List Using `foreach`

- Working lists of arbitrary items (type `do foreach_list`)

```
foreach item in this that and the other {
        display "I found something: `item'"
}
foreach cmd in describe codebook summarize {
        `cmd' mpg
}
```

- The results show how each list is processed one item at a time

```
. foreach item in this that and the other {
  2.         display "I found something: `item'"
  3. }
I found something: this
I found something: that
I found something: and
I found something: the
I found something: other
. foreach cmd in describe codebook summarize {
  2.         `cmd' mpg
  3. }
              storage  display     value
variable name   type   format      label      variable label
------------------------------------------------------------------------------
mpg             int    %8.0g                  Mileage (mpg)
------------------------------------------------------------------------------
mpg                                                           Mileage (mpg)
------------------------------------------------------------------------------
                  type:  numeric (int)
                 range:  [12,41]                      units:  1
         unique values:  21                       missing .:  0/74
                  mean:   21.2973
```

```
                    std. dev:    5.7855
                  percentiles:       10%       25%       50%       75%       90%
                                      14        18        20        25        29

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
         mpg |        74     21.2973    5.785503        12         41

.
```

Working a Variable List Using `foreach`

- Working with a *varlist* (`do foreach_var`)

```
foreach var of varlist mpg-turn {
   local theLab : var lab `var´
   disp `"The variable label of `var´ is ->`theLab´<-"´
}
```

- Stata understands to expand the *varlist* before processing it

```
. foreach var of varlist mpg-turn {
  2.    local theLab : var lab `var´
  3.    disp `"The variable label of `var´ is ->`theLab´<-"´
  4. }
The variable label of mpg is ->Mileage (mpg)<-
The variable label of rep78 is ->Repair Record 1978<-
The variable label of headroom is ->Headroom (in.)<-
The variable label of trunk is ->Trunk space (cu. ft.)<-
The variable label of weight is ->Weight (lbs.)<-
The variable label of length is ->Length (in.)<-
The variable label of turn is ->Turn Circle (ft.) <-
```

Notes on `foreach`

- When working with an arbitrary list, the preposition is `in`. When working with a specially named item, the preposition is `of`
- The loop is echoed to the screen (with line-numbering), but the contents of the loop are not echoed for each pass through the loop

Example Using `forvalues`

- Here are some silly computations (`do forval_simple`)

```
forvalues cnt=1/5 {
        display "The square root of `cnt' is " sqrt(`cnt')
}
```

- The output is predictable

```
. forvalues cnt=1/5 {
  2.         display "The square root of `cnt´ is " sqrt(`cnt´)
  3. }
The square root of 1 is 1
The square root of 2 is 1.4142136
The square root of 3 is 1.7320508
The square root of 4 is 2
The square root of 5 is 2.236068
```

Building a Graph with `forvalues`

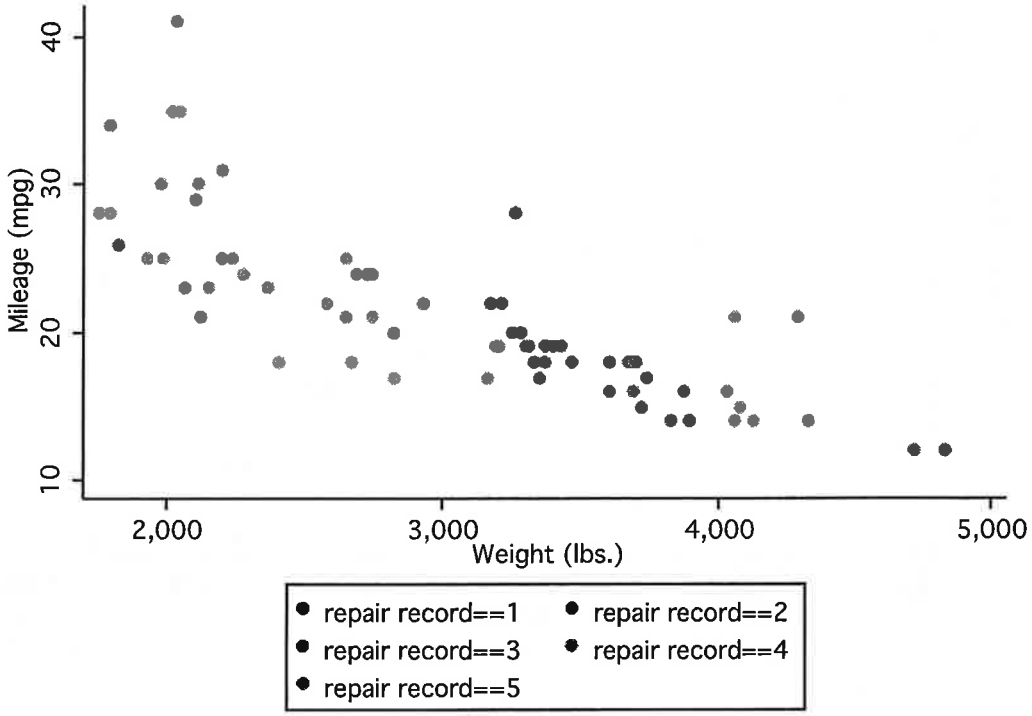
- This code builds multiple overlays for each value of `rep78` (`do forval_gph`)

```
forvalues which=1/5 {
        local plot "`plot' (scatter mpg weight if rep78==`which')"
        local legend `"`legend' `which' "repair record==`which'""'
}
// to include missing values we would need to add them similarly:
//   local plot "`plot' (scatter mpg weight if rep78==., mlabel(rep78))"
display `"`legend'"'
twoway `plot', legend(order(`legend'))
graph save forval_gph, replace
display "The actual graph command was "
display as input `"twoway `plot', legend(order(`legend'))"'
```

- The command builds a complicated graph command

```
. forvalues which=1/5 {
  2.         local plot "`plot´ (scatter mpg weight if rep78==`which´)"
  3.         local legend `"`legend´ `which´ "repair record==`which´""´
  4. }
. // to include missing values we would need to add them similarly:
. //   local plot "`plot´ (scatter mpg weight if rep78==., mlabel(rep78))"
. display `"`legend´"´
 1 "repair record==1" 2 "repair record==2" 3 "repair record==3" 4 "repair recor
> d==4" 5 "repair record==5"
. twoway `plot´, legend(order(`legend´))
. graph save forval_gph, replace
(file forval_gph.gph saved)
. display "The actual graph command was "
The actual graph command was
. display as input `"twoway `plot´, legend(order(`legend´))"´
twoway  (scatter mpg weight if rep78==1) (scatter mpg weight if rep78==2) (scat
> ter mpg weight if rep78==3) (scatter mpg weight if rep78==4) (scatter mpg wei
> ght if rep78==5), legend(order( 1 "repair record==1" 2 "repair record==2" 3 "
> repair record==3" 4 "repair record==4" 5 "repair record==5"))
```

The Graph Itself



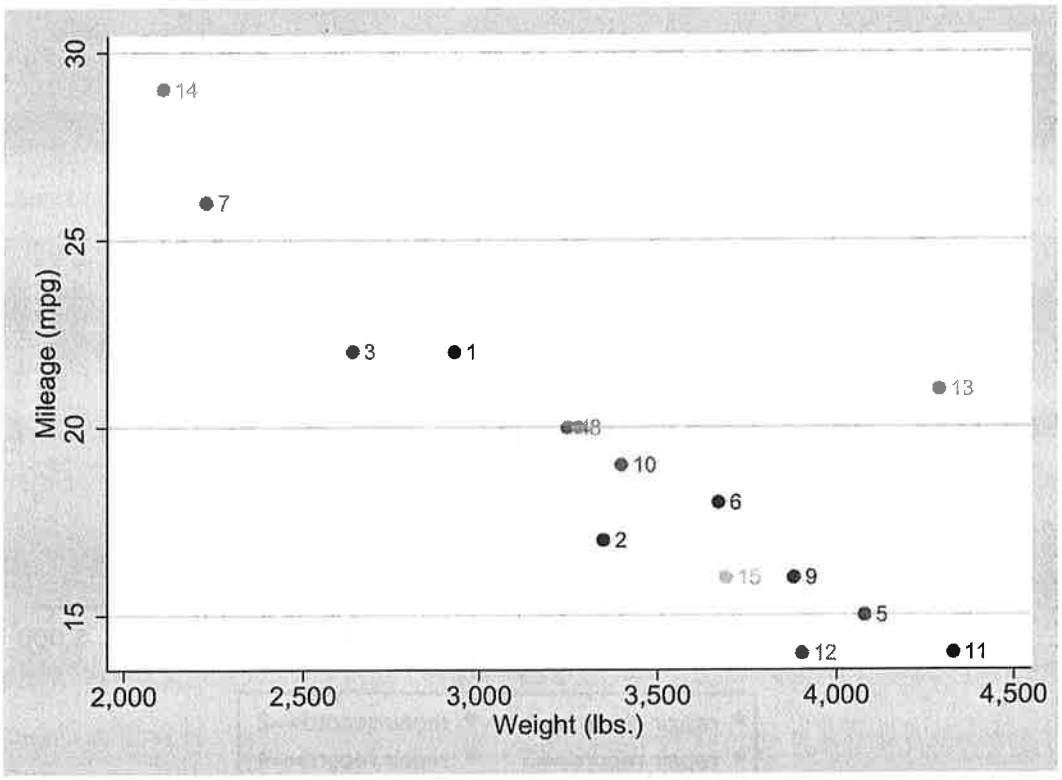
A More Complex Build

- Here is another loop which shows the different symbols made by the default scheme, and then by the `s2mono` scheme:

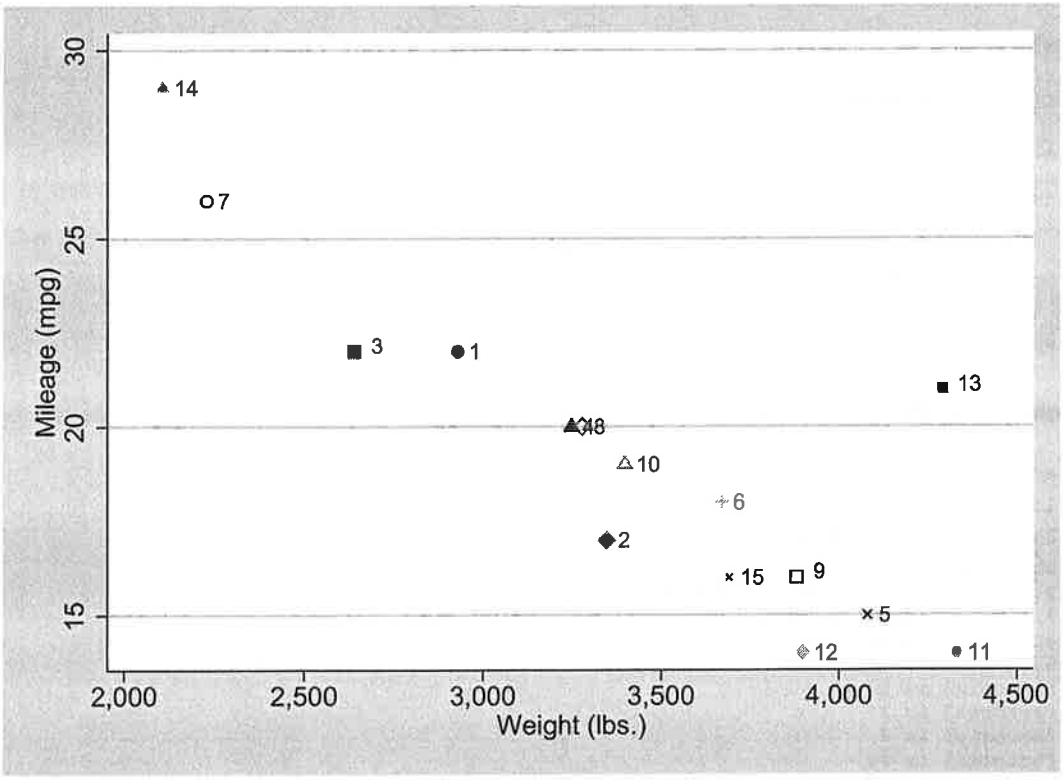
```
* example of a tempvar
tempvar obsnum
gen `obsnum' = _n
forvalues which=1/15 {
        local plot "`plot' (scatter mpg weight if _n==`which', mlabel(`obsnum')
> )"
}
twoway `plot', legend(off)
display "The actual graph command was "
display as input "twoway `plot', legend(off)"
graph save "s2color_points", replace
more
* now with a different scheme:
twoway `plot', legend(off) scheme(s2mono)
graph save "s2mono_points", replace
```

- Building commands using loops is quite useful

The `s2color` points



The `s2mono` points



Notes on Curly Braces

- In Stata, nothing except a space or a comment can follow curly braces ({ and })
  - This is not much of a limitation—it is mentioned because it might surprise some people

---

`while`

- Stata also has a `while` loop:

```
while exp {
    something to do
    }
```

- The loop will keep running until the *exp* evaluates to 0
  - ...because 0 is the only thing which is False in Stata

Example of `while`

- This shows some simple computations (type `do while_ex`

```
scalar FACT = 1
local cnt = 1
local toomuch 1000000
while FACT <= `toomuch´ {
        scalar FACT = FACT * `cnt´
        display "`cnt´ factorial is " FACT
        local ++cnt
}
display "Finally past `toomuch´!"
```

- Factorials get big fast!

```
. scalar FACT = 1
. local cnt = 1
. local toomuch 1000000
. while FACT <= `toomuch´ {
  2.          scalar FACT = FACT * `cnt´
  3.          display "`cnt´ factorial is " FACT
  4.          local ++cnt
  5. }
1 factorial is 1
2 factorial is 2
3 factorial is 6
4 factorial is 24
5 factorial is 120
6 factorial is 720
7 factorial is 5040
8 factorial is 40320
9 factorial is 362880
10 factorial is 3628800
. display "Finally past `toomuch´!"
Finally past 1000000!
```

Getting Out of Loop

- The `continue` command will cause Stata to jump to the next iteration of the loop
- Using `continue, break` will break out of the loop entirely

15.3.2 Conditionals

The `if` Command

- Stata has an `if` command (which is very different than the `if` qualifier!)
- The `if` command is for executing code conditional on a statement.
- We'll look at help for the syntax

```
help ifcmd
```

`if` Example

- Dropping every variable in `auto` that doesn't begin with `m` (type `do if_ex`

```
foreach var of varlist _all {
        if substr("`var'",1,1)!="m" {
                drop `var'
        }
}
```

- Look in the results window to see what is left...

Other Branching

- Stata does not have `goto`-style branching in its regular ado language.
- This is not a limitation...

15.4 Conclusion

15.4.1 A Few Final Comments

Working in the Command window

- Believe it or not, all of the above can be done from Stata's command window
- If you find yourself typing loops into the Command window, stop right away, and use the Do-file editor.

Finishing Up

- Close your log file
