Sociology 7708: Hierarchical Linear Modeling **Instructor: Natasha Sarkisian**

Introduction to Hierarchical Linear Modeling

Hierarchical models (also known as multilevel models or mixed models) are used to handle nested data structures, e.g.:
Students – classrooms – schools
Workers – firms
Individuals – neighborhoods
Households – countries
Repeated measures over time – individuals

Traditional regression methods only allow analyzing such data by focusing on one level – e.g. focusing on students and ignoring that they are nested within classrooms and schools – but that leads to errors.

Using relationships between variables on the level of the individuals to make conclusions about groups → atomistic fallacy
[Note: We can aggregate the information collected from individuals to characterize groups – but the analysis should be conducted on the level of groups.]

Using relationships between variables describing groups to make conclusions about relationships between variables on the level of individuals → ecological fallacy

Traditional regression methods assume that the relationships among individual-level variables are the same in all groups (they assume homogeneity of regression). Therefore, they do not allow to assess how the relationships between variables at the individual level might vary according to the group context, or to study what it is about group context that may cause such variation.

In addition, traditional regression methods produce biased results in multilevel samples – they assume independent observations, and therefore misestimate standard errors – fail to take into account the dependence among observations that belong to the same group.

HLM regression models resolve all these problems:

- They allow simultaneously estimating relationships at individual and group levels
- They allow for the relationships at the individual level to vary across groups
- They allow to model cross-level interactions – i.e., examine how group-level factors shape individual-level relationships
- They allow to correctly estimate standard errors by dividing the unexplained variance into two components – group-level (random effects of groups) and individual level: $Y_{ij} = \alpha + X\beta + u_j + e_{ij}$ where u_j is the effect of membership in group j, and e_{ij} is the residual effect for individual i within this group.

Such model with random effects can also be interpreted as a two-level model, where:

Level 1 model is: $Y_{ij} = \alpha + X\beta + e_{ij}$
Level 2 model is: $\alpha = \mu + u_i$,

In this kind of model, we treat the intercept α as a random variable. This is called conditional model with a random intercept. We will learn more about this and other types of HLM models in this course. The basic point is: When studying complex, multilevel processes, we should use theories and analytic techniques that are also multilevel – hence we need HLM.

Introduction to Stata

Preparation for Using Stata on Citrix

STEP 1. Use your browser to go to BC Apps on bcapps.bc.edu, login using your BC credentials, and open Stata 17SE app. For more information on troubleshooting, see https://www.bc.edu/offices/help/teaching/app_server.html

Note: There are times when too many people use Stata 17, and the system doesn't have enough licenses to add one more person; you can use Stata 16 instead in these cases.

STEP 2. To be able to store files on BC server, map the drive for BC apps storage; the detailed instructions for each operating system are available here: http://www.bc.edu/offices/help/teaching/app_server/apps-files/files-map.html. You have to be at BC or connect using BC VPN (https://www.bc.edu/offices/help/getstarted/network/vpn.html) in order to have access to BC apps storage.

Note: On Macs, you might need to repeat mapping the drive steps from time to time if BC Apps storage does not show up in a list of available folders. So keep the link and remember how to do this step for potential future use if you have a Mac.

If you run into problems, contact the BC technology help center. If they cannot resolve your issue or you need further help, please come see me.

STEP 3. Open your BC apps storage folder and create a subfolder for this class; call your subfolder socy7708.

STEP 4. Download GSS 2012 dataset from the course webpage and place it into your socy7708 subfolder in your folder on BC apps storage server.

STEP 5. Once again, go to BC Apps on bcapps.bc.edu and open Stata 17SE app, or go back to that window if it's still open. Open GSS 2012 file from inside Stata program by typing the following commands in the command line on the bottom of Stata screen (press enter after each command line):

```
cd “L:\socy7708\”
log using stataprep.log, replace
use gss2012.dta, clear
log close
```

All of these commands should run and generate no red error messages on your Stata screen if all went well. If that's the case, you can close Stata and BC Apps once you are done.

STEP 6. Open a file explorer program and navigate to the BC Apps storage folder. Find stataprep.log file in your list and open it using any text editor. Make sure it contains a few lines of text, including your "use gss2012.dta, clear" command. Close it again.

STEP 7. Open your email, attach the stataprep.log file (make sure to navigate to the apps storage folder and then to socy7708 subfolder) and email it to yourself.

Opening data files

Download GSS 2012 dataset from the course webpage; place it into your folder on BC apps storage server. Open Stata 17SE on BCApps.

To check the current working directory, type pwd in the Command window immediately after starting Stata (without running a cd command).

```
. pwd
L:\
```

Let's change the working directory for easier file access using cd (c=change d=directory) command (alternatively, you can specify the path each time, or open and save files using Stata menus):

```
. cd "L:\socy7708\"
```

Now that we have the correct working directory set up, we can open the data file from inside Stata program using the following command:

```
. use gss2012.dta, clear
```

You can also open files from the web, e.g.:

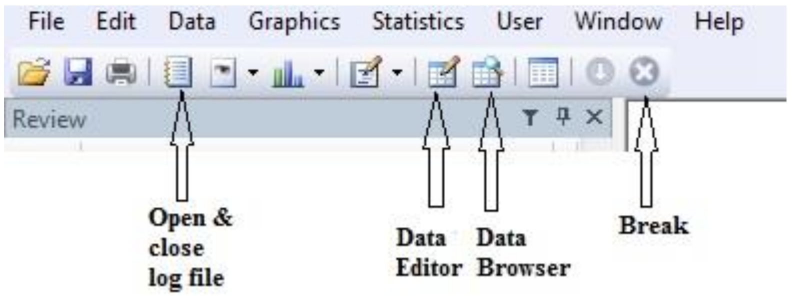
```
. use https://www.sarkisian.net/socy7708/gss2012.dta, clear
```

Keeping a record of your work

Opening the log file (we include replace option so that you don't get an error if you already used that log file name before – it will get replaced by the new one; if you want to add to the existing log, use "append" instead of "replace):

```
log using learn_stata.log, replace
```

To see the log, you can at any time press the button and view a snapshot of the log. (You can also close or suspend log using that same button.)



Two types of log -- .log and .scml. I choose .log rather than .scml type of file so it can be read in any text editor or word processor. I would recommend that you always use .log format for now. But you can also easily convert .scml type log into the text format log:

```
translate mylog.smcl mylog.log
```

You can also use translate command to recover a log when you have forgotten to start one:

```
translate @Results mylog.txt
```

Note that if you are opening a Stata log file in a Word processor, you should change the font to a fixed width font, such as Courier New (otherwise the output looks misaligned). Courier New 10 or 9 point usually works the best. Otherwise things won't be aligned.

Basic syntax of Stata commands:

1. Command – What do you want to do?
2. Names of variables, files, etc. – Which variables or files do you want to use?
3. Qualifier on observations -- Which observations do you want to use?
4. Options – Do you have any other preferences regarding this command?

Help and installation

Help in Stata – help and search commands:

```
. help tabulate

. search logistic

Keyword search
        Keywords:  logistic
          Search:  (1) Official help files, FAQs, Examples, SJs, and STBs

Search of official help files, FAQs, Examples, SJs, and STBs
[U]     Chapter 26  . . . . . . . . .  Overview of Stata estimation commands
        (help estcom)

[R]     clogit  . . . . . .  Conditional (fixed-effects) logistic regression
        (help clogit)

[R]     cloglog . . . . . . . . . . . . . . . Complementary log-log regression
        (help cloglog)

[R]     constraint  . . . . . . . . . . . . . . .  Define and list constraints
        (help constraint)

[R]     fracpoly  . . . . . . . . . . . . . . Fractional polynomial regression
        (help fracpoly)

[R]     glogit  . . . . . . . . . . . . . .  Logit and probit for grouped data
        (help glogit)

[R]     logistic  . . . . . . . . . Logistic regression, reporting odds ratios
        (help logistic)

[R]     logistic postestimation . . . . . .  Postestimation tools for logistic
        (help logistic postestimation)

[R]     logit . . . . . . . . . .  logistic regression, reporting coefficients
        (help logit)

[R]     logit postestimation  . . . . . . . . . Postestimation tools for logit
        (help logit postestimation)
```

```
[R]     mfp . . . . . . . . . . . . Multivariable fractional polynomial models
        (help mfp)

[R]     mlogit  . . . . . . . . . Multinomial (polytomous) logistic regression
        (help mlogit)

[R]     nlogit  . . . . . . . . . . . . . . . . . . . .  Nested logit regression
        (help nlogit)

[R]     ologit  . . . . . . . . . . . . . . . . . .  Ordered logistic regression
        (help ologit)
--Break--
r(1);
```

You can also use "net search" command that will search Stata resources online in addition to local resources:

```
. net search spost
(contacting http://www.stata.com)

16 packages found (Stata Journal and STB listed first)
-----------------------------------------------------
st0094 from http://www.stata-journal.com/software/sj5-4
    SJ5-4 st0094.  Confidence intervals for predicted outcomes... / Confidence
    intervals for predicted outcomes in regression / models for categorical
    outcomes / by Jun Xu and J. Scott Long, Indiana University / Support:
    spostsup@indiana.edu / After installation, type help prvalue and prgen

spost9_ado from http://www.indiana.edu/~jslsoc/stata
    spost9_ado | Stata 9-13 commands for the post-estimation interpretation /
    Distribution-date: 05Aug2013 / of regression models. Use package
    spostado.pkg for Stata 8. / Based on Long & Freese - Regression Models for
    Categorical Dependent / Variables Using Stata. Second Edition. / Support

spost9_do from http://www.indiana.edu/~jslsoc/stata
    spost9_do | SPost9 example do files. / Distribution-date: 27Jul2005 / Long
    & Freese 2005 Regression for Categorical Dependent Variables / using
    Stata. Second Edition. Stata Version 9. / Support
    www.indiana.edu/~jslsoc/spost.htm / Scott Long & Jeremy Freese

spostado from http://www.indiana.edu/~jslsoc/stata
    spostado: Stata 8 commands for the post-estimation interpretation of /
    regression models. Based on Long's Regression Models for Categorical / and
    Limited Dependent Variables. / Support: www.indiana.edu/~jslsoc/spost.htm
    / Scott Long & Jeremy Freese (spostsup@indiana.edu)

spostrm7 from http://www.indiana.edu/~jslsoc/stata
    spostrm7: Stata 7 do & data files to reproduce RM4CLDVs results using
    SPost. / Files correspond to chapters of Long: Regression Models for
    Categorical / & Limited Dependent Variables. / Support:
    www.indiana.edu/~jslsoc/spost.htm / Scott Long & Jeremy Freese

spostst8 from http://www.indiana.edu/~jslsoc/stata
    spostst8: Stata 8 do & data files to reproduce RM4STATA results using
    SPost. / Files correspond to chapters of Long & Freese-Regression Models
    for Categorical / Dependent Variables Using Stata (Stata 8 Revised
    Edition). / Support: www.indiana.edu/~jslsoc/spost.htm / Scott Long &
```

```
spost13_ado from http://www.indiana.edu/~jslsoc/stata
    Distribution-date: 15Jul2015 / spost13_ado | SPost13 commands from Long
    and Freese (2014) / Regression Models for Categorical Outcomes using
    Stata, 3rd Edition. / Support www.indiana.edu/~jslsoc/spost.htm / Scott
    Long (jslong@indiana.edu) & Jeremy Freese (jfreese@northwestern.edu)

spost9_legacy from http://www.indiana.edu/~jslsoc/stata
    Distribution-date: 18Feb2014 / spost9_legacy | SPost9 commands not
    included in spost13_ado. / From Long and Freese, 2014, Regression Models
    for Categorical Outcomes / using Stata, 3rd Edition. / Support
    www.indiana.edu/~jslsoc/spost.htm / Scott Long (jslong@indiana.edu) &

spost13_do from http://www.indiana.edu/~jslsoc/stata
    Distribution-date: 05Aug2014 / spost13_do | SPost13 examples from Long and
    Freese, 2014, / Regression Models for Categorical Outcomes using Stata,
    3rd Edition. / Support www.indiana.edu/~jslsoc/spost.htm / Scott Long
    (jslong@indiana.edu) & Jeremy Freese (jfreese@northwestern.edu)

spost13_do12 from http://www.indiana.edu/~jslsoc/stata
    Distribution-date: 11Aug2014 / spost13_do12 | SPost13 examples for Stata
    12 from Long and Freese, 2014, / Regression Models for Categorical
    Outcomes using Stata, 3rd Edition. / Support
    www.indiana.edu/~jslsoc/spost.htm / Scott Long (jslong@indiana.edu) &

difd from http://fmwww.bc.edu/RePEc/bocode/d
    'DIFD': module to evaluate test items for differential item functioning
    (DIF) / DIF detection is a first step in assessing bias in test items.  /
    difd detects DIF in test items between groups, conditional on / the trait
    that the test is measuring, using logistic / regression.  The criteria for

difdetect from http://fmwww.bc.edu/RePEc/bocode/d
    'DIFDETECT': module to detect and adjust for differential item functioning
    (DIF) / Detection of and adjustment for differential item functioning /
    (DIF):  Identifies differential item functioning, creates / dummy/virtual
    items to be used to adjust ability (trait) / estimates, and calculates the

difwithpar from http://fmwww.bc.edu/RePEc/bocode/d
    'DIFWITHPAR': module for detection of and adjustment for differential item
    functioning (DIF) / Identifies differential item functioning, creates /
    dummy/virtual items to be used to adjust ability (trait) / estimates in
    PARSCALE, writes the code and data file needed to / process the updated

grcompare from http://fmwww.bc.edu/RePEc/bocode/g
    'GRCOMPARE': module to make group comparisons in binary regression models
    / This is a Stata module to make group comparisons in binary / regression
    models using alternative measures, including gradip: / average difference
    in predicted probabilities over a range; / grdiame:difference in group

prepar from http://fmwww.bc.edu/RePEc/bocode/p
    'PREPAR': module to write code and data file needed to process variables
    in PARSCALE / This program writes the input code and data file for
    PARSCALE, / which is a real time-saver if you aren't familiar with /
    PARSCALE.  / KW: PARSCALE / Requires: Stata version 8.2, PARSCALE and

runparscale from http://fmwww.bc.edu/RePEc/bocode/r
    'RUNPARSCALE': module to run PARSCALE from Stata / Builds a PARSCALE data
    file and command file, executes the / command file, displays the PARSCALE
    log file in Stata results / window, and merges the PARSCALE theta
    estimates and their / standard errors back into the original data set.  /
```

```
1 reference found in tables of contents
---------------------------------------
http://www.indiana.edu/~jslsoc/stata/
    2014-08-10 / SPost:  Interpreting regression models. Scott Long & Jeremy
    Freese / Workflow:  Workflow of data analysis. Scott Long / Teaching:
    Teaching files. Scott Long / Research:  Research examples & commands.
    Scott Long / Support:  www.indiana.edu/~jslsoc/spost.htm /
```

Note that some of the things we found are user-written programs that implement user-written commands that can be quite helpful; to install, click on the package and click to install, or type

```
. net install spost13_ado, from(http://www.indiana.edu/~jslsoc/stata)
```

Also, if you have Stata on your own computer, do not forget to do Stata updates on a regular basis, including updating all installed programs (ado files).

```
. update all
```

Good resource for learning Stata:
http://www.ats.ucla.edu/stat/stata/

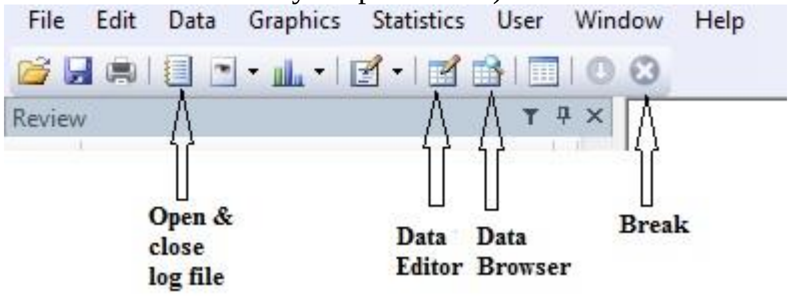
Forum to ask questions about Stata (but search for answers first!):
http://www.statalist.org/

Examining and editing the data

Describing the dataset:

```
. des

Contains data from L:\socy7708\gss2012.dta
  obs:         1,974
 vars:           800                          11 Sep 2013 06:50
 size:     1,717,380
-------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
-------------------------------------------------------------------------------
year            int     %8.0g                 GSS YEAR FOR THIS RESPONDENT
id              int     %8.0g                 RESPONDNT ID NUMBER
wtss            double  %12.0g     WTSS       WEIGHT VARIABLE
vpsu            byte    %8.0g      LABA       Variance primary sampling unit
vstrat          int     %8.0g      LABA       Variance stratum
abany           byte    %8.0g      LABB       ABORTION IF WOMAN WANTS FOR ANY
                                                REASON
abdefect        byte    %8.0g      LABB       STRONG CHANCE OF SERIOUS DEFECT
abhlth          byte    %8.0g      LABB       WOMANS HEALTH SERIOUSLY
                                                ENDANGERED
abnomore        byte    %8.0g      LABB       MARRIED--WANTS NO MORE CHILDREN
abpoor          byte    %8.0g      LABB       LOW INCOME--CANT AFFORD MORE
                                                CHILDREN
abrape          byte    %8.0g      LABB       PREGNANT AS RESULT OF RAPE
absingle        byte    %8.0g      LABB       NOT MARRIED
accntsci        byte    %8.0g      LABC       HOW SCIENTIFIC: ACCOUNTING
accptoth        byte    %8.0g      LABD       R ACCEPT OTHERS EVEN WHEN THEY DO
                                                THINGS WRONG
acqntsex        byte    %8.0g      ACQNTSEX   R HAD SEX WITH ACQUAINTANCE LAST
                                                YEAR
actupset        byte    %8.0g      LABE       PPL AT WORK THROW THINGS WHEN
                                                UPSET WITH R
--Break--
r(1);
```

I used Break button to stop Stata from producing more output. If you do want to see all the output, either click on the more link on the bottom of the output viewer, or press space key. For some of you, more link doesn't appear and the output appears all at once rather than one page at a time. That is regulated with "set more off" and "set more on" commands in Stata (you can add perm option to make Stata remember your preference).



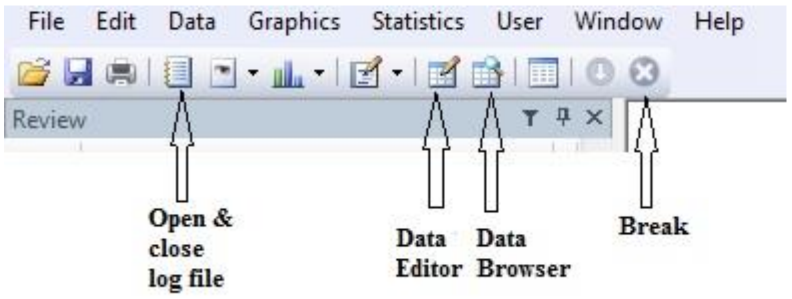
Get codebook info:

```
. codebook class
-------------------------------------------------------------------------------
class                                          SUBJECTIVE CLASS IDENTIFICATION
-------------------------------------------------------------------------------

                  type:  numeric (byte)
                 label:  CLASS

                 range:  [1,4]                        units:  1
         unique values:  4                        missing .:  17/1974

            tabulation:  Freq.   Numeric  Label
                           200         1  LOWER CLASS
                           853         2  WORKING CLASS
                           839         3  MIDDLE CLASS
                            65         4  UPPER CLASS
                            17         .
```

List values of selected variables for each observation:

```
. list  wrkstat hrs1 wrkslf

      +----------------------------+
      |  wrkstat   hrs1     wrkslf |
      |----------------------------|
   1. |  WORKING     15    SOMEONE |
   2. |  WORKING     30    SOMEONE |
   3. |  WORKING     60    SOMEONE |
   4. |    other      .    SOMEONE |
   5. |  retired      .    SOMEONE |
      |----------------------------|
   6. |    other      .    SOMEONE |
   7. |  KEEPING      .    SOMEONE |
  --Break--
r(1);
```

Using data browser to look at the data and data editor to change data:



There is no UNDO button!!! That applies to all data management commands, too. But the changes are made only to the dataset that's in Stata's memory. So if you close it without saving, you can always start again with your original dataset. If, however, you want to save your changes, you should save the data file with any changes you made – typically with a different filename:

```
. save gss2012changed.dta, replace
file L:\socy7708\gss2012changed.dta saved
```

If you are not sure you want to keep your changes, use "preserve" command in the beginning to save a copy of the dataset in Stata memory; restore in the end will return the data to that saved version.

Using do-files
You should keep a do-file with all your analysis steps – that way, if you make a mistake, you can easily rerun things. To have that, we can save all the commands that we did interactively into a do-file, or we can right away write a do-file and then execute it. Open do-file editor, create and save your file (.do) – or use doedit command. You can execute that file from the do-file editor or using the command line:

```
. do mydofile.do
```

But be careful to specify the location of your file or make sure it is in the working directory specified in the last "cd" command.

It is often convenient to create and edit do-files in another text editor – in Windows, I prefer TextPad: http://www.textpad.com; another good option is Notepad++. For a Mac OS, you can use Sublime Text.

And if you want to save all commands you've done so far, right click on the command window and select "Save Review Contents." If some of your commands had errors (highlighted in red), you can right click on each of them and delete them from the Review window before copying. Or you can select some commands and send them to do editor by right clicking and selecting Send to Do File Editor.

You can also keep the log of just the commands:

```
cmdlog using filename
```

Then you can use that log as a do-file.

It's a good idea to specify Stata version in the beginning of each do file, e.g.

```
version 17
```

Whether in do files or when entering commands interactively, it is useful to include comments on what you are doing: Everything typed after a star (*) or after // is treated as a comment and not executed; same with any text between /* and */

In addition, people often use /// as a line break tool to better format do-files:

```
use gss2012.dta, clear
sum age /// here I am summarizing age
wrkstat /// here I am summarizing work status, and next sex
sex
```

Note that you can't include /// on the last line of a command (or in the end of a one-line command) because otherwise it doesn't see a carriage return and doesn't execute that command at all. Use star to create comments on a separate line in such cases.
Lines in do files can be either separated with line breaks (CR=carriage return) or a delimiter ;
To change, you specify the following command in the beginning of a do file.
#delimit ;

To restore the carriage return delimiter inside a file, use #delimit cr. When a do-file begins execution, the delimiter is automatically set to carriage return, even if it was called from another do-file that set the delimiter to semicolon. Also, the current do-file need not worry about restoring the delimiter to what it was because Stata will do that automatically.

Closing log and exiting Stata

```
. log close
. exit, clear
```

Descriptive Statistics in Stata

Let's reopen the data file and continue our log:

```
. use gss2012.dta, clear
. log using learn_stata.log, append
```

Frequency tables -- tabulate command:

```
. tab class
   SUBJECTIVE |
        CLASS |
IDENTIFICATIO |
            N |      Freq.     Percent        Cum.
--------------+-----------------------------------
  LOWER CLASS |        200       10.22       10.22
WORKING CLASS |        853       43.59       53.81
 MIDDLE CLASS |        839       42.87       96.68
  UPPER CLASS |         65        3.32      100.00
--------------+-----------------------------------
        Total |      1,957      100.00
```

This also allows us to identify the mode – here, WORKING CLASS is the mode.

Including missing values:

```
. tab class, miss

   SUBJECTIVE |
        CLASS |
IDENTIFICATIO |
            N |      Freq.     Percent        Cum.
---------------+-----------------------------------
  LOWER CLASS |        200       10.13       10.13
WORKING CLASS |        853       43.21       53.34
 MIDDLE CLASS |        839       42.50       95.85
  UPPER CLASS |         65        3.29       99.14
            . |         17        0.86      100.00
---------------+-----------------------------------
        Total |      1,974      100.00
```

To suppress labels and see numeric values:

```
. tab class, nol

 SUBJECTIVE |
      CLASS |
IDENTIFICAT |
        ION |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |        200       10.22       10.22
          2 |        853       43.59       53.81
          3 |        839       42.87       96.68
          4 |         65        3.32      100.00
------------+-----------------------------------
      Total |      1,957      100.00
```

Multiple univariate tables of frequencies are obtained using tab1 command:

```
. tab1 marital class

-> tabulation of marital

      MARITAL |
       STATUS |      Freq.     Percent        Cum.
--------------+-----------------------------------
      married |        900       45.59       45.59
      widowed |        163        8.26       53.85
     divorced |        317       16.06       69.91
    separated |         68        3.44       73.35
NEVER MARRIED |        526       26.65      100.00
--------------+-----------------------------------
        Total |      1,974      100.00

-> tabulation of class

   SUBJECTIVE |
        CLASS |
IDENTIFICATIO |
            N |      Freq.     Percent        Cum.
--------------+-----------------------------------
  LOWER CLASS |        200       10.22       10.22
WORKING CLASS |        853       43.59       53.81
 MIDDLE CLASS |        839       42.87       96.68
  UPPER CLASS |         65        3.32      100.00
--------------+-----------------------------------
        Total |      1,957      100.00
```

Measures of central tendency and variability:

```
. sum tvhours
    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
     tvhours |      1298    3.088598      2.8651          0         24


. sum tvhours, detail
                  HOURS PER DAY WATCHING TV
```

```
-------------------------------------------------------------
      Percentiles      Smallest
 1%            0              0
 5%            0              0
10%            1              0       Obs                1298
25%            1              0       Sum of Wgt.        1298

50%            2                      Mean           3.088598
                        Largest       Std. Dev.        2.8651
75%            4             24
90%            6             24       Variance       8.208798
95%            8             24       Skewness       3.123997
99%           15             24       Kurtosis       18.48296

. tabstat tvhours, stats(mean median min max p25 p75 range iqr sd variance)
    variable |      mean       p50       min       max       p25       p75
-------------+------------------------------------------------------------
     tvhours |  3.088598         2         0        24         1         4
--------------------------------------------------------------------------

    variable |     range       iqr        sd  variance
-------------+----------------------------------------
     tvhours |        24         3    2.8651  8.208798
------------------------------------------------------
```

Cross-tabulation:

```
. tab  wrkslf wrkgovt

R SELF-EMP OR |    GOVT OR PRIVATE
    WORKS FOR |       EMPLOYEE
     SOMEBODY | governmen    private |     Total
--------------+----------------------+----------
SELF-EMPLOYED |         6        161 |       167
 SOMEONE ELSE |       365      1,324 |     1,689
--------------+----------------------+----------
        Total |       371      1,485 |     1,856
```

With column percentages:

```
. tab  wrkslf wrkgovt, col

+-------------------+
| Key               |
|-------------------|
|     frequency     |
| column percentage |
+-------------------+

R SELF-EMP OR |    GOVT OR PRIVATE
    WORKS FOR |       EMPLOYEE
     SOMEBODY | governmen    private |     Total
--------------+----------------------+----------
SELF-EMPLOYED |         6        161 |       167
              |      1.62      10.84 |      9.00
--------------+----------------------+----------
 SOMEONE ELSE |       365      1,324 |     1,689
              |     98.38      89.16 |     91.00
--------------+----------------------+----------
        Total |       371      1,485 |     1,856
              |    100.00     100.00 |    100.00
```

With three types of percentages and chi-square test:

```
. tab  wrkslf wrkgovt, col row cell chi2
```

```
+-------------------+
| Key               |
|-------------------|
|     frequency     |
|  row percentage   |
| column percentage |
|  cell percentage  |
+-------------------+

R SELF-EMP OR |    GOVT OR PRIVATE
    WORKS FOR |       EMPLOYEE
     SOMEBODY | governmen    private |     Total
--------------+----------------------+----------
SELF-EMPLOYED |         6        161 |       167 
              |      3.59      96.41 |    100.00 
              |      1.62      10.84 |      9.00 
              |      0.32       8.67 |      9.00 
--------------+----------------------+----------
 SOMEONE ELSE |       365      1,324 |     1,689 
              |     21.61      78.39 |    100.00 
              |     98.38      89.16 |     91.00 
              |     19.67      71.34 |     91.00 
--------------+----------------------+----------
        Total |       371      1,485 |     1,856 
              |     19.99      80.01 |    100.00 
              |    100.00     100.00 |    100.00 
              |     19.99      80.01 |    100.00 

          Pearson chi2(1) =  30.8474   Pr = 0.000
```

Data Management in Stata

Creating new variables: typically done using gen (often followed by replace), recode, or egen commands; egen is more advanced and we won't cover it for now, but do look at that command and its options if you are doing data management in Stata.

For example, I want to create a variable that is 0 for those people who work less than 40 hours a week and 1 for those who work 40 hours a week or more (we call this a dichotomy or a dummy variable, where 0 means absence of some characteristic and 1 means presence). Let's first examine the variable that exists in the data set:

```
. codebook hrs1
-------------------------------------------------------------------------------
hrs1                                               NUMBER OF HOURS WORKED LAST WEEK
-------------------------------------------------------------------------------

                  type:  numeric (byte)
                 label:  LABAD, but 68 nonmissing values are not labeled

                 range:  [1,89]                       units:  1
         unique values:  68                       missing .:  808/1,974

              examples:  40
                         45
                         .
                         .

. sum hrs1, det
              NUMBER OF HOURS WORKED LAST WEEK
-------------------------------------------------------------
```

```
      Percentiles      Smallest
 1%            5              1
 5%            9              1
10%           19              2       Obs               1,166
25%           34              4       Sum of Wgt.       1,166

50%           40                      Mean           40.27358
                        Largest       Std. Dev.      15.54011
75%           48             89
90%           60             89       Variance        241.495
95%           65             89       Skewness       .0575558
99%           80             89       Kurtosis       3.649988
```

We start by generating a new variable called hrs40 with all missing values. Then we will first fill in zeroes for those who work less than 40 hours for pay, and finally fill in 1s for those who work 40+ hours. To do so, we need to use conditions expressed as an "if" statement. To express conditions, we can use the following:
< less
> more
== equal
<= less or equal
>= more or equal
~= or != not equal
Can connect them with & (and) and | (or).
Can also use parentheses to combine conditions.

So in our case, we do:

```
. gen hrs40=.
(1,974 missing values generated)

. replace hrs40 = 0 if hrs1<40
(377 real changes made)

. replace hrs40 = 1 if hrs1>=40 & hrs1~=.
(789 real changes made)

. tab hrs40, missing
      hrs40 |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |        377       19.10       19.10
          1 |        789       39.97       59.07
          . |        808       40.93      100.00
------------+-----------------------------------
      Total |      1,974      100.00
```

Label the variable:

```
. label variable hrs40 "R works 40 hours a week or more"
```

Label its values: two steps, first define a set of labels, then apply this set to a variable:

```
. label define hrs40label 0 "less than 40" 1 "40 or more"
. label values hrs40 hrs40label

. tab hrs40, missing
  R works 40 |
hours a week |
     or more |      Freq.     Percent        Cum.
-------------+-----------------------------------
less than 40 |        377       19.10       19.10
  40 or more |        789       39.97       59.07
           . |        808       40.93      100.00
-------------+-----------------------------------
```

```
      Total |      1,974      100.00

. codebook hrs40
-------------------------------------------------------------------------------
hrs40                                                R works 40 hours a week or more
-------------------------------------------------------------------------------
                  type:  numeric (float)
                 label:  hrs40label

                 range:  [0,1]                        units:  1
         unique values:  2                        missing .:  808/1,974

            tabulation:  Freq.   Numeric  Label
                           377         0  less than 40
                           789         1  40 or more
                           808         .
```

To rename a variable:

```
. rename hrs40 hours40
```

There is a simpler way to generate a dummy variable that only uses one step; for example, to generate a dichotomy indicating married respondents (0=not married, 1=married):

```
. codebook marital
-------------------------------------------------------------------------------
marital                                                            MARITAL STATUS
-------------------------------------------------------------------------------
                  type:  numeric (byte)
                 label:  MARITAL

                 range:  [1,5]                        units:  1
         unique values:  5                        missing .:  0/1,974

            tabulation:  Freq.   Numeric  Label
                           900         1  married
                           163         2  widowed
                           317         3  divorced
                            68         4  separated
                           526         5  NEVER MARRIED

. gen married=(marital==1) if marital<.

. tab married
    married |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |      1,074       54.41       54.41
          1 |        900       45.59      100.00
------------+-----------------------------------
      Total |      1,974      100.00
```

The additional condition, if marital<. , is included in order to preserve missing data – without that condition, anyone with marital variable missing would be coded as 0. Here, however, we don't have any missing values on marital status.

Another command we can use to create variables is recode command. For example, to generate marital status with 3 categories, married, previously married, never married:

```
. recode marital (1=1) (2/4=2) (5=3), gen(marital3)
(1249 differences between marital and marital3)
. tab marital3
  RECODE of |
    marital |
   (MARITAL |
    STATUS) |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |        900       45.59       45.59
          2 |        548       27.76       73.35
          3 |        526       26.65      100.00
------------+-----------------------------------
      Total |      1,974      100.00
```

Label the new variable:

```
. label variable marital3 "marital status 3 categories"
```

Label values of the new variable:

```
. label define marital3label 1 "married" 2 "previously married" 3 "never married"
. label values marital3 marital3label
```

Check the results:

```
. codebook marital3
-------------------------------------------------------------------------------------------
marital3                                                     marital status 3 categories
-------------------------------------------------------------------------------------------
                  type:  numeric (byte)
                 label:  marital3label

                 range:  [1,3]                        units:  1
         unique values:  3                        missing .:  0/1,974

            tabulation:  Freq.   Numeric  Label
                           900         1  married
                           548         2  previously married
                           526         3  never married
```

These are various recoding techniques to create and deal with categorical and dichotomous variables; but recoding is also very useful for continuous variables. For example, they often have extreme values (outliers) that could impact our results unduly, and to reduce their impact, we can topcode or bottomcode the variable, that is, recode the very high or very low values to bring them more into the regular range. For example:

```
. tab agekdbrn, miss

    R'S AGE |
   WHEN 1ST |
 CHILD BORN |      Freq.     Percent        Cum.
------------+-----------------------------------
         13 |          1        0.05        0.05
         14 |          7        0.35        0.41
         15 |         20        1.01        1.42
         16 |         30        1.52        2.94
         17 |         68        3.44        6.38
         18 |        101        5.12       11.50
         19 |        102        5.17       16.67
         20 |        110        5.57       22.24
         21 |        134        6.79       29.03
         22 |         78        3.95       32.98
         23 |         95        4.81       37.79
         24 |         82        4.15       41.95
         25 |         94        4.76       46.71
         26 |         70        3.55       50.25
         27 |         78        3.95       54.20
         28 |         58        2.94       57.14
         29 |         52        2.63       59.78
         30 |         55        2.79       62.56
         31 |         32        1.62       64.18
         32 |         32        1.62       65.81
         33 |         28        1.42       67.22
         34 |         29        1.47       68.69
         35 |         21        1.06       69.76
         36 |         13        0.66       70.42
         37 |         11        0.56       70.97
         38 |         10        0.51       71.48
         39 |          7        0.35       71.83
         40 |          6        0.30       72.14
         41 |          3        0.15       72.29
         42 |          1        0.05       72.34
         46 |          1        0.05       72.39
         50 |          1        0.05       72.44
          . |        544       27.56      100.00
------------+-----------------------------------
      Total |      1,974      100.00
```

There are many missing values here as well; those are people without children. The extreme values are probably legitimate values, but in many cases, if using this variable, you might still want to bottomcode and topcode it (i.e., assign all values below 13 to have the value of 13 and all values above 44, you would assign value 44) in order to avoid the undue influence of extreme values on your results. Here, I will be generating a top-coded and bottom-coded version of agekdbrn; only the most extreme outliers (<1% of distribution) are typically top and bottom coded:

```
. . gen agekdbrn_tb=clip(agekdbrn, 13, 44)
(544 missing values generated)

. sum agekdbrn_tb
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
 agekdbrn_tb |      1,430    24.07413    5.589537         13         44

. gen agekdbrn_t=clip(agekdbrn, ., 44)
(544 missing values generated)

. sum agekdbrn_t
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
  agekdbrn_t |      1,430    24.07413    5.589537         13         44
```

Note that missing values are still coded as missing, they are not topcoded into the top value.

You can also create new variables by transforming existing continuous variables. The mathematical operators can be used: addition +, subtraction -, multiplication *, division /, and powers ^. You can also use parentheses when calculations are complex and the order of operations matters.

Age squared:
```
. gen age2=age^2
(5 missing values generated)
```

Square root of age:
```
. gen agesqrt=sqrt(age)
(5 missing values generated)
```

Log of age:
```
. gen agelg=log(age)
(5 missing values generated)
```

Saving the dataset with the newly created variables:
```
. save gss2012_2.dta
file L:\socy7708\gss2012_2.dta saved
```

Entering your own data

We can do that in data editor, but we need to clear the memory first:
```
clear all
```

Then you can open the editor and start entering numbers. If you have variables that contain text, you can just enter text; the resulting variables will be non-numeric (string). If, however, that text only includes a few categories, then you should assign numbers to those categories and enter numbers. For example, if you have a variable "year in college," don't type "junior" or "senior" but enter numbers 1, 2, 3, or 4 and then label the values using label define and label values commands, as discussed above.

In addition to entering the data themselves, you should also name the variables either in the Variables window (bottom right) or using commands, e.g.
```
rename var1 income
```

When done, save the new dataset:

```
. save enter_example.dta
file L:\socy7708\enter_example.dta saved
```

Graphics in Stata

```
. scatter hrs1 prestg80

. graph matrix hrs1 hrs2 prestg80 sphrs1 sppres80

. graph bar, over(class) blabel(bar)

. histogram hrs1, normal
(bin=32, start=1, width=2.75)
```

We can save graphs for future use (in Stata-specific graph format; can only be opened in Stata):
```
graph save mygraph.gph
```

To then display that graph, we type:
```
graph use mygraph.gph
```

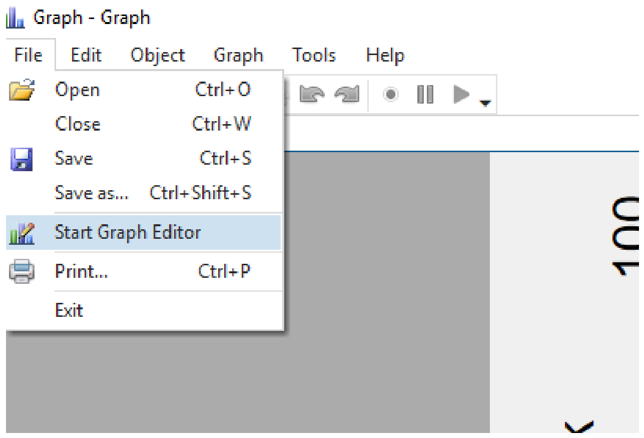
You can also export them into different, non-Stata formats:

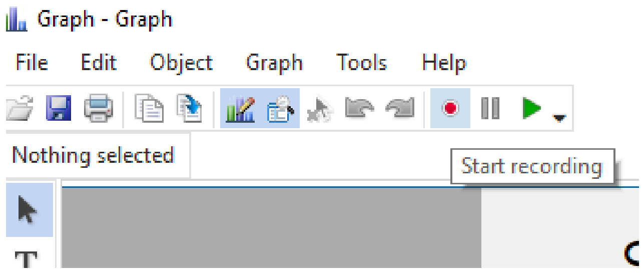
```
. graph export mygraph.wmf, replace
```

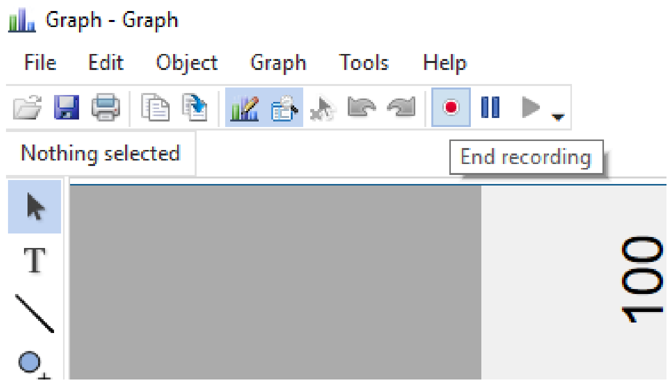
The output format is determined by the suffix of the file name (see help graph export):
```
                  Implied
suffix            option            Output format
-------------------------------------------------------------------
.ps               as(ps)            PostScript
.eps              as(eps)           EPS (Encapsulated PostScript)
.wmf              as(wmf)           Windows Metafile
.emf              as(emf)           Windows Enhanced Metafile
.pdf              as(pdf)           PDF
.png              as(png)           PNG (Portable Network Graphics)
.tif              as(tif)           TIFF
```

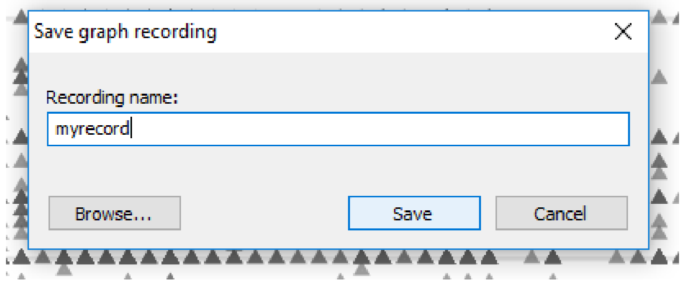
WMF works best for Windows computers, PNG works best for Macs.

You can also edit graphs in interactive mode using Graph Editor in Stata; however, I would recommend using options to generate the graphs you need because that allows for reproduceability. There is, however, also an option to use Recorder together with Graph Editor – it's not quite the same level of reproducibility and readability as using syntax, but at least then you can reapply all the changes to another graph (or another version of the same graph) – after starting Graph Editor, press Record prior to editing the graph, then press stop when you are done and give your recording a name:









Recording name: myrecord

You can then create another graph and apply a recording to it:

```
. scatter hrs1 occ80, play(myrecord)
```

To further explore the options available for graphics, use:

```
. help graph
```

Stata versions and settings

There are different versions of Stata: Variable number limits are 2,047 for Stata/IC, and 99 for Small Stata. When using Stata/MP and Stata/SE, the maximum number of variables in your dataset can be changed by using “set maxvar” command. The default value of maxvar is 5,000 for Stata/MP and Stata/SE. Here, we are using Stata/IC; the version on the apps server is Stata/SE.

Besides set maxvar, to make it easier for you to work with Stata, you can change some of other default settings using “set” command, e.g.:

```
        set logtype text
        set more off
```

Some Stata settings can be made “permanent” -- for example, if you want Stata to never pause output with a --more-- in the Results window, you could type

```
. set more off, perm
```

Another useful set command that you will likely encounter once you start running statistical models on large data is “set matsize” (can also be used with ”permanently” option). set matsize sets the maximum number of rows/columns that can be placed in a matrix used by Stata's data analysis commands.

For Stata/IC, the initial value is 400, but it may be changed upward or downward. The upper limit is 800. For Stata/MP and Stata/SE, the default value is 400, but it may be changed upward or downward. The upper limit is 11,000. This command may not be used with Small Stata; matsize is permanently frozen at 100.

Another useful set command has to do with graphs.

```
.  set scheme schemename [, permanently]
```

set scheme allows you to set the graphics scheme to be used. The default setting is s2color. You can use point and click to explore graphics schemes.