14.170: Programming for Economists

1/12/2009-1/16/2009

Melissa Dell
Matt Notowidigdo
Paul Schrimpf

What is 14.170?

- 6.170 is a very popular undergraduate course at MIT titled "Introduction to Software Engineering." Goals of the course are threefold:
  1. Develop good programming habits
  2. Learn how to implement basic algorithms
  3. Learn various specific features and details of a popular programming language (currently Java, but has been Python, Scheme, C, C++ in the past)

- We created a one-week course 14.170 with similar goals:
  1. Develop good programming habits
  2. Learn how to implement basic algorithms
  3. Learn various specific features and details of several popular programming language (Stata, Perl, Matlab, C)

- Course information
  - **COURSE WEBPAGE: http://web.mit.edu/econ-gea/14.170**
  - E-mails ([at] mit dot edu):
    - mdell
    - noto
    - paul_s

COURSE OVERVIEW

Today (MATT): Basic Stata, Intermediate Stata, MLE and NLLS in Stata

Tuesday (MATT): Mata, GMM, Large data sets, numerical precision issues

Wednesday (PAUL): Basic, Intermediate and Advanced Matlab

Thursday (MATT, MELISSA): Perl, GIS

Friday (PAUL): Intro to C, More C (C + Matlab, C + Stata)

(see syllabus for more details)

Lecture 1, Basic Stata

Basic Stata overview slide

- Basic data management
  - Reading, writing data sets
  - Generating, re-coding, parsing variables (+ regular expressions, if time is permitting)
  - Built-in functions
  - Sorting, merging, reshaping, collapsing

- Programming language details (control structures, variables, procedures)
  - forvalues, foreach, while, if, in
  - Global, local, and temporary variables
  - Missing variables (*worst* programming language design decision in all of Stata)

- Programming “best practices”
  - Comments!
  - Assertions
  - Summaries, tabulations (and LOOK at them!)

- Commonly-used built-in features
  - Regression and post-estimation commands
  - Outputting results

Data Management

- The key manual is "**Stata Data Management**"
- You should know almost every command in the book very well before you prepare a data set for a project
- Avoid re-inventing the wheel
- We will go over the most commonly needed commands (but we will not go over all of them)
- Type "help *command*" to find out more in Stata, e.g. "help infile"
- Standard RA "prepare data set" project
  1. Read in data
  2. Effectively summarize/tabulate data, present graphs
  3. Prepare data set for analysis (generate, reshape, parse, encode, recode)
  4. Preliminary regressions and output results

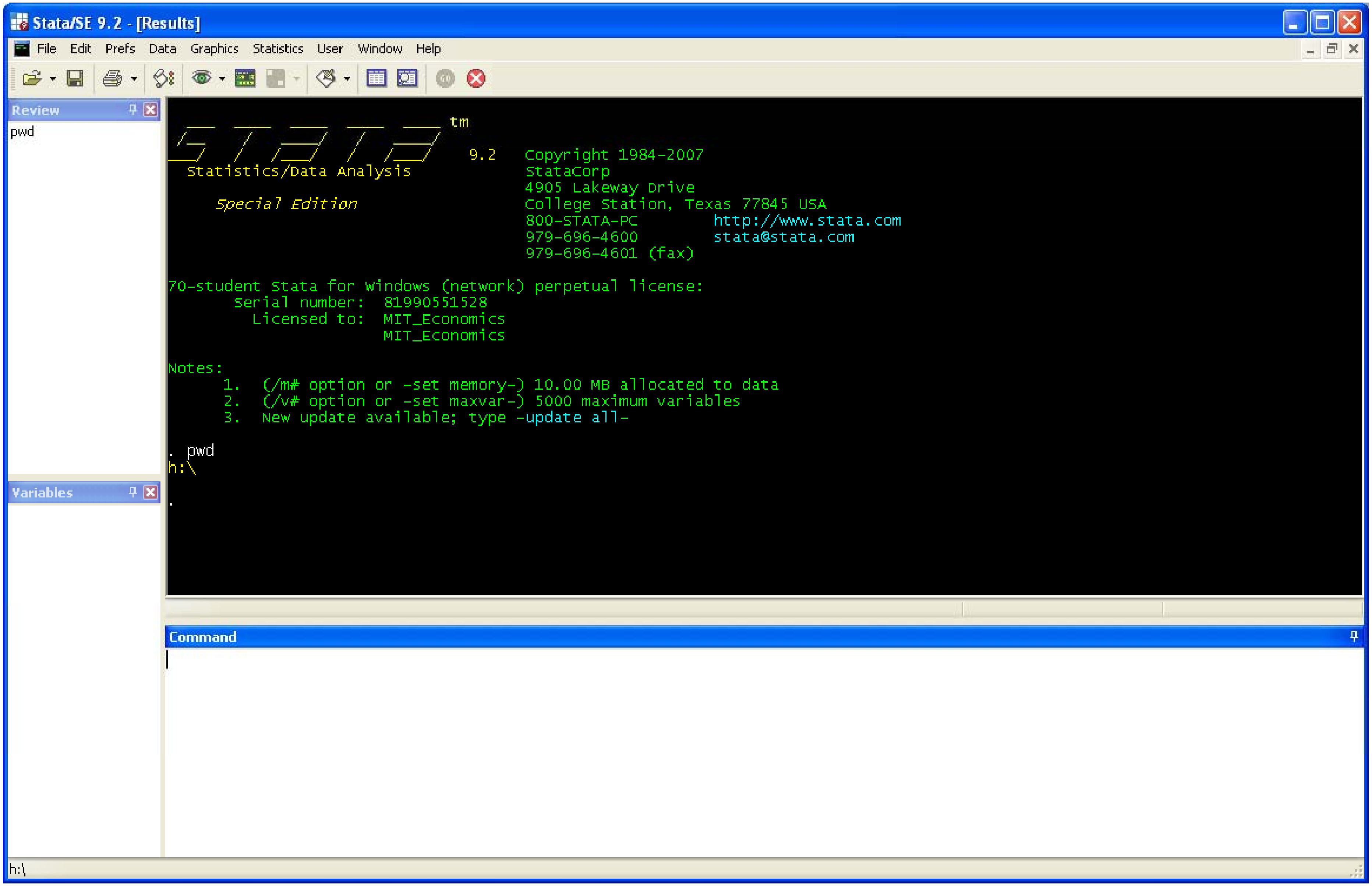
Getting started

- There are several ways to use Stata …

	DO file editor	
operating system	Stata user interface	Text editor (e.g. emacs, TextPad)
Windows	**(A)**	**(B)**
UNIX / Linux	(C)	**(D)**

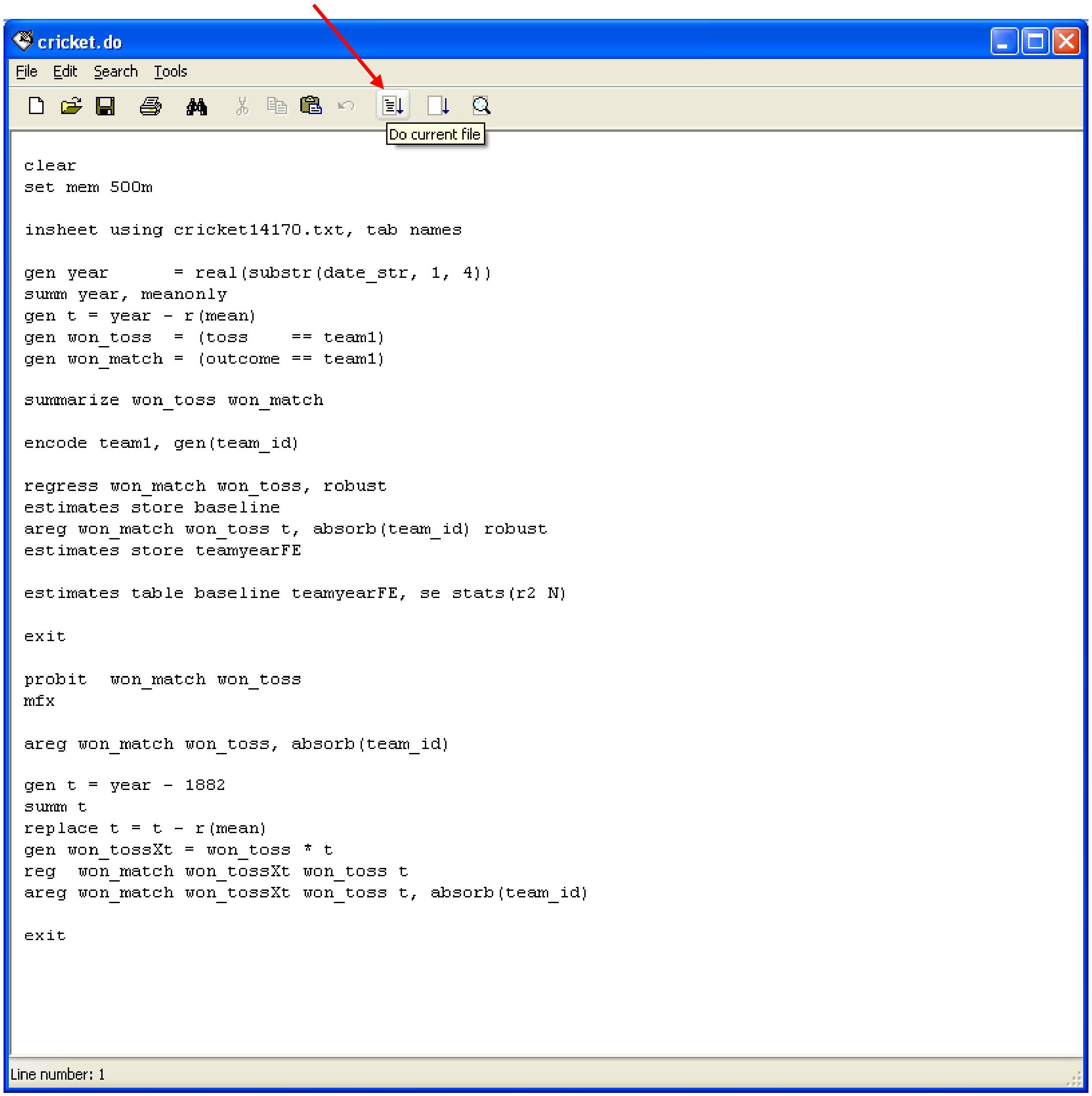
- I recommend starting with (A)
- I use (D) because I find the emacs text editor to be very effective (and conducive to good programming practice)

Getting started, (A)



Getting started, (A), con’t

Press
“Ctrl-8”
to open editor!



Reading in data

If data is already in Stata file format (thanks NBER!), we are all set …

```
clear
set memory 500m
use "/proj/matt/aha80.dta"
```

If data is not in Stata format, then can use insheet for tab-delimited files or infile or infix for fixed-width files (with or without a data dictionary). Another good option is to use Stat/Transfer

```
clear
set memory 500m
insheet using "/proj/matt/cricket/data.txt", tab
```

```
clear
set memory 500m
infix ///
 int     year           1-4 ///
 byte    statefip       14-15 ///
 byte    sex            30 ///
 byte    hrswork        53-54 ///
 long    incwage        62-67 ///
using cps.dat
```

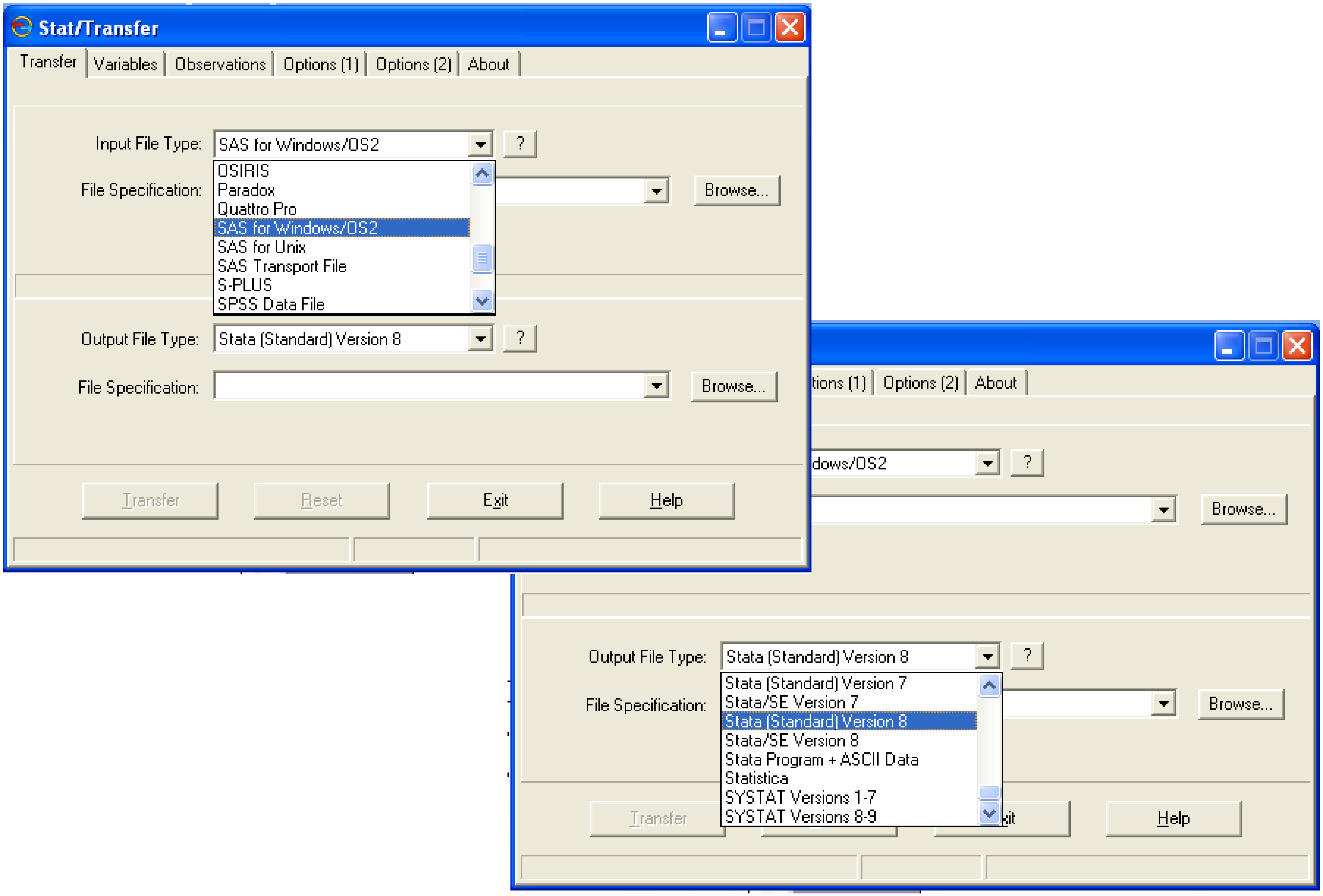
insheet data

player	year	round	pick	position	height	weight	season	salary	estimated
Andrew Bogut	**2005**	**1st**	**1st**	**C**	**84**	**245**	**2006**	**4340520**	**0**
Marvin Williams	2005	1st	2nd	F	81	230	2006	3883560	0
Deron Williams	2005	1st	3rd	G	75	210	2006	3487400	0
Chris Paul	**2005**	**1st**	**4th**	**G**	**75**	**175**	**2006**	**3144240**	**0**
Raymond Felton	2005	1st	5th	G	73	198	2006	2847360	0
Martell Webster	2005	1st	6th	G-F	81	210	2006	2586120	0
Charlie Villanueva	2005	1st	7th	F	83	240	2006	2360880	0
Channing Frye	2005	1st	8th	F-C	83	248	2006	2162880	0
Ike Diogu	2005	1st	9th	F	80	250	2006	1988160	0
Andrew Bynum	2005	1st	10th	C	84	285	2006	1888680	0
Yaroslav Korolev	2005	1st	12th	F	81	203	2006	1704480	0
Sean May	2005	1st	13th	F	81	266	2006	1619280	0
Rashad McCants	2005	1st	14th	G	76	207	2006	1538400	0
Antoine Wright	2005	1st	15th	G-F	79	210	2006	1461360	0
Joey Graham	2005	1st	16th	G-F	79	225	2006	1388400	0
Danny Granger	2005	1st	17th	F	80	225	2006	1318920	0
Gerald Green	2005	1st	18th	F	80	200	2006	1253040	0
Hakim Warrick	2005	1st	19th	F	81	219	2006	1196520	0
Julius Hodge	2005	1st	20th	G	79	210	2006	1148760	0

infix data

196502513581**09**0025135801016**6**11001	11003341	**00**0002488**000000**
196502513581**09**00251358020158**2**1001	11003102	**40**0002180**002000**
196502513581**09**00265891030116**2**1006	13105222	**00**0000030**000000**
196502513581**09**00323843030134**1**1006	15007102	**40**0007259**005250**
19650251358110900251358010165 11001	01001341	000005000005000
196502513581109002513580201552 1001	10003102	400004200004200
196502464591109002464590101561 1001	11003102	540004500004500
196502464591109002464590201532 1001	12004311	000000000000000
196502464591109002228200301181 1006	14106331	000000000000000
196502563361109002563360101681 1001	06002212	000005067004827
196502563361109002563360201602 1001	10103311	000000000000000
196502207511109002207510101471 2001	10003212	000002100002100
196502207511109002207510201432 2001	13005102	232002000002000

Stat/Transfer



Describing and summarizing data

```
describe
summarize
list in 1/100
list if exptot > 1000000 | paytot > 1000000

summarize exptot paytot, detail
tabulate ctscnhos, missing
tabulate cclabhos, missing
```

Stata data types

id	**str7**	%9s	A.H.A identification number
reg	**byte**	%8.0g	region code
stcd	byte	%8.0g	state code
hospno	str4	%9s	hospital number
ohsurg82	byte	%8.0g	open heart surgery
nerosurg	byte	%8.0g	neurosurgery
bdtot	**long**	%12.0g	beds set up
admtot	**double**	%10.0g	total admissions
ipdtot	double	%10.0g	total inpatient days

Stata data types, con't

- Good programming practices:
  - Choose the right data type for your data ("admissions" is a double?)
  - Choose good variable names ("state_code", "beds_total", "region_code")
  - Make the values intuitive (open heart surgery should be 0/1 dummy variable, not either 1 or 5, where 5 means "hospital performs open heart surgery")
- Stata details:
  - String data types can be up to 244 characters (why 244?)
  - Decimal variables are "float" by default, NOT "double"
    - "float" variables have ~7 decimal places of accuracy while "double" variables have ~15 decimal places of accuracy (floats are 4 bytes of data, doubles are 8 bytes of data).
    - When is this important? MLE, GMM. Variables that are used as "tolerances" of search routines should <u>always</u> be double. We will revisit this in lecture 3. In general, though, this distinction is not important.
    - If you are paranoid (like me!), can place "**set type double, permanently**" at top of your file and all decimals will be "double" by default (instead of "float")

Summarizing data

Variable	Obs	Mean	Std. Dev.	Min	Max
id	0				
reg	7201	5.024024	2.460094	0	9
stcd	7201	53.80475	24.42879	2	95
hospno	0				
dtbeg	6420	80722.21	28295.46	10179	123179
dbegm	6420	8.052181	2.829625	1	12
dbegd	6420	1.213396	2.18991	1	31
dbegy	6420	79.05888	.2380478	78	80
dtend	6420	85535.24	20171.31	10580	123180
dendm	6420	8.245327	2.015929	1	12
dendd	6420	30.02009	1.966395	1	31
dendy	6420	79.95717	.2100529	79	81
dcov	7201	365.1984	11.82492	58	370
fyr	6420	1.005763	.0757028	1	2
fisyr	6385	61091.62	35947	10179	123180

- Why only 6420 observations for “fyr” variable? 0 observations for “id” variable?
- Are there any missing “id” variables? How could we tell?
- How many observations are in the data set?

Missing data in Stata

(<u>Disclaimer:</u> In my opinion, this is one the worst "features" of Stata. It is counter-intuitive and error-prone. But if you use Stata you are stuck with their bad programming language design. So learn the details!)

- Missing values in Stata
  - Missing numeric values are represented as a "." (a period). Missing string values are "" (an empty string of length 0)
  - Best way to think about "." value: it is "+/- infinity" (it is an unattainably large or an unattainably small number that no valid real number can equal).

    generate `c =` **log**`(0)` produces <u>only</u> missing values

  - What might be wrong with following code?

    ```
    drop if weeks_worked < 40
    regress log_wages is_female is_black age education_years
    ```

- Missing values in Stata, new "feature" starting in version 9.1: 27 missing values!
  - Now missing values can be ".", ".a", … , ".z"
  - If "." is infinity, then ".a" is infinity+1
  - For example, to drop ALL possible missing values, you need to write code like this:

    ```
    drop if age >= .
    ```

  - <u>Cannot</u> be sure in recent data sets (especially government data sets that feel the need to use new programming features) that "`drop if age == .`" will drop ALL missing age values
  - Best programming practice (in Stata 9):

    ```
    drop if missing(age)
    ```

Detailed data summaries

```
clear
set mem 100m
set obs 50000
generate normal  = invnormal(uniform())
generate ttail30 = invttail(30, uniform())
generate ttailX  = invttail(5+floor(25*uniform()), uniform())
summ normal ttail* , detail
```

```
                          normal
-------------------------------------------------------------
      Percentiles      Smallest
 1%    -2.372783      -4.460569
 5%    -1.638569      -4.451361
10%    -1.283004       -4.45108       Obs               50000
25%    -.6750032      -4.124427       Sum of Wgt.       50000

50%     .0005758                      Mean          -.000249
                        Largest       Std. Dev.     1.003072
75%     .6831857       3.802239
90%     1.283304       3.856395       Variance      1.006154
95%     1.636142       3.928157       Skewness     -.0213135
99%     2.309629       4.377341       Kurtosis      3.009086

                          ttail30
-------------------------------------------------------------
      Percentiles      Smallest
 1%    -2.443862      -4.734029
 5%    -1.691493      -4.629606
10%     -1.30687      -4.592993       Obs               50000
25%    -.6884176       -4.27448       Sum of Wgt.       50000

50%    -.0046165                      Mean         -.0002087
                        Largest       Std. Dev.     1.032878
75%     .6841348       4.236295
90%     1.321023       4.294293       Variance      1.066838
95%     1.695416       4.294732       Skewness      .0128472
99%      2.45744        4.70623       Kurtosis      3.167685
```

```
                          ttailX
-------------------------------------------------------------
      Percentiles      Smallest
 1%    -2.635827      -9.313776
 5%    -1.772502      -7.945157
10%    -1.346782      -7.466501       Obs               50000
25%    -.6922761      -6.730944       Sum of Wgt.       50000

50%     .0022048                      Mean          .0047822
                        Largest       Std. Dev.     1.097506
75%     .6990915        7.36639
90%     1.356226       7.775033       Variance       1.20452
95%     1.773723       8.391249       Skewness      .0625497
99%     2.690659       13.99303       Kurtosis      4.665932
```

leptokurtic distribution!

Tabulating data

```
clear
set obs 1000
generate c =
  log(floor(10*uniform()))
tabulate c, missing
```

c	Freq.	Percent	Cum.
0	109	10.90	10.90
.6931472	108	10.80	21.70
1.098612	93	9.30	31.00
1.386294	76	7.60	38.60
1.609438	105	10.50	49.10
1.791759	92	9.20	58.30
1.94591	117	11.70	70.00
2.079442	106	10.60	80.60
2.197225	98	9.80	90.40
.	96	9.60	100.00
Total	1,000	100.00	

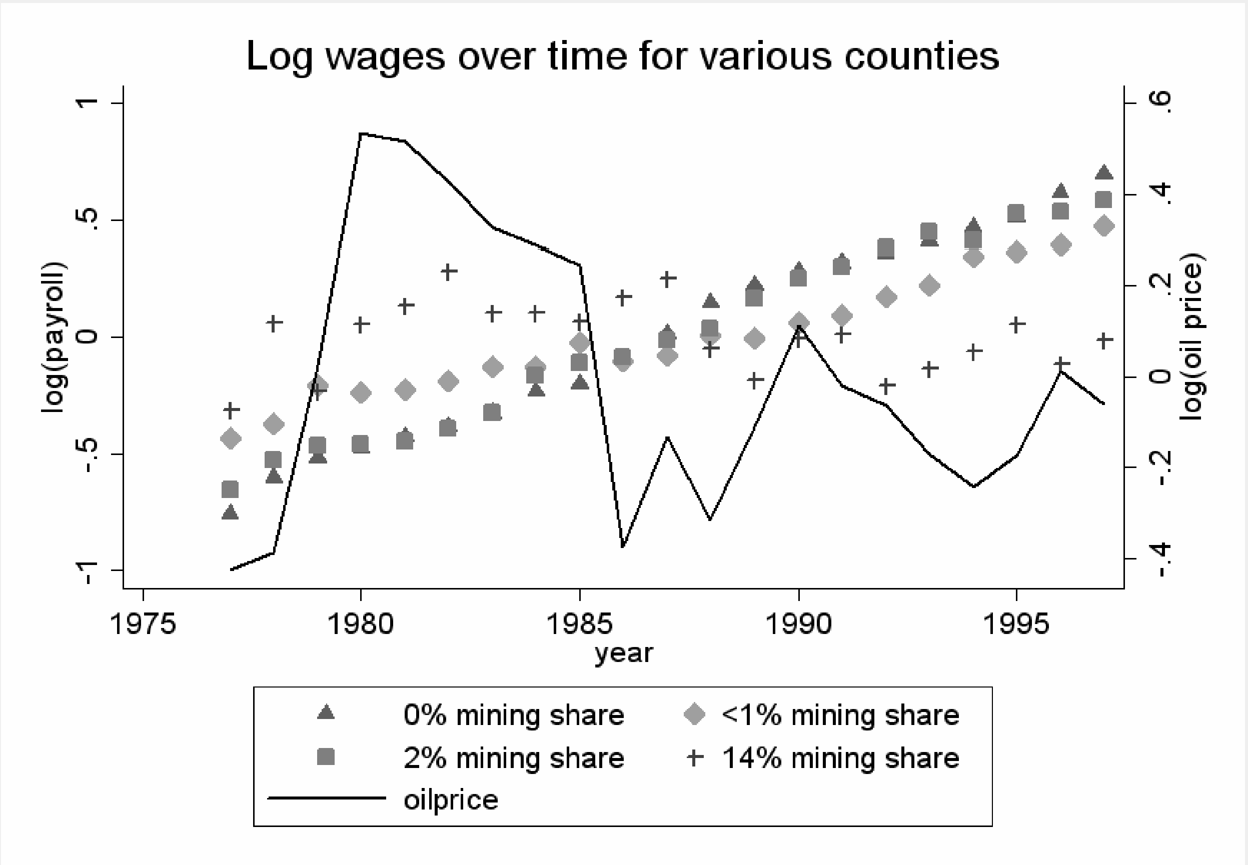
Two-way tables

```
clear
set obs 10000
generate rand = uniform()
generate cos = round( cos(0.25 * _pi * ceil(16 * rand)), 0.0001)
generate sin = round( sin(0.25 * _pi * ceil(16 * rand)), 0.0001)
tabulate cos sin, missing
```

	sin					
cos	-1	-.7071	0	.7071	1	Total
-1	0	0	1,261	0	0	1,261
-.7071	0	1,287	0	1,213	0	2,500
0	1,249	0	0	0	1,238	2,487
.7071	0	1,234	0	1,284	0	2,518
1	0	0	1,234	0	0	1,234
Total	1,249	2,521	2,495	2,497	1,238	10,000

Presenting data graphically

- Type “help twoway” to see what Stata has built-in
  - Scatterplot
  - Line plot (connected and unconnceted)
  - Histogram
  - Kernel density
  - Bar plot
  - Range plot



Preparing data for analysis

- Key commands:
  - generate
  - replace
  - if, in
  - sort, gsort
  - merge
  - reshape
  - by
  - egen
    - count
    - diff
    - group
    - max
    - mean
    - median
    - min
    - mode
    - pctile
    - rank
    - sd
    - rowmean, rowmax, rowmin
  - encode
  - assert
  - count
  - append
  - collapse
  - strfun
    - length
    - lower
    - proper
    - real
    - regexm, regexr
    - strpos
    - subinstr
    - substr
    - trim
    - upper

De-meaning variables

```
clear
set obs 1000
generate variable  = log(floor(10*uniform()))
summ variable
replace variable = variable - r(mean)
summ variable
```

```
. summ variable

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
    variable |       899    1.413929     .695583          0   2.197225

. replace variable = variable - r(mean)
(899 real changes made)

. summ variable

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
    variable |       899    5.96e-09     .695583  -1.413929   .7832957
```

NOTE: “infinity” – r(mean) = “infinity”

De-meaning variables, con't

```
clear
set obs 1000
generate variable  = log(floor(10*uniform()))
egen variable_mean = mean(variable)
replace variable = variable - variable_mean
summm variable
```

```
. summ variable

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
    variable |       889    1.423383    .6757646          0   2.197225

. egen variable_mean = mean(variable)

. replace variable = variable - variable_mean
(889 real changes made)

. summ variable

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
    variable |       889    4.17e-08    .6757646  -1.423383    .7738413
```

if/in commands

```
clear
set obs 50000
generate normal = invnormal(uniform())
list in 1/5
list in -5/-1
generate two_sigma_event = 0
replace two_sigma_event = 1 if (abs(normal)>2.00)
tabulate two_sigma_event
```

```
. list in 1/5

     +-----------+
     |    normal |
     |-----------|
  1. | -1.139813 |
  2. | -.2582426 |
  3. | -1.418245 |
  4. |  -.798243 |
  5. |  1.266219 |
     +-----------+

. list in -5/-1

       +-----------+
       |    normal |
       |-----------|
49996. | -.2370036 |
49997. |  .0296064 |
49998. |  1.883285 |
49999. |  1.259045 |
50000. | -.3780237 |
       +-----------+

. generate two_sigma_event = 0

. replace  two_sigma_event = 1 if (abs(normal)>2.00)
(2253 real changes made)

. tabulate two_sigma_event

two_sigma_e |
       vent |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |     47,747       95.49       95.49
          1 |      2,253        4.51      100.00
------------+-----------------------------------
      Total |     50,000      100.00
```

egen commands

Calculate denominator of logit log-likelihood function …

```
egen double denom = sum(exp(theta))
```

Calculate 90-10 log-income ratio …

```
egen inc90 = pctile(inc), p(90)
egen inc10 = pctile(inc), p(10)
gen  log_90_10 = log(inc90) – log(inc10)
```

Create state id from 1..50 (why would we do this?) …

```
egen group_id = group(state_string)
```

Make sure all income sources are non-missing …

```
egen any_income_missing = rowmiss(inc*)
replace any_income_mising = (any_income_missing > 0)
```

by, sort, gsort

```
clear
set obs 1000

** randomly generate states (1-50)
gen state = 1+floor(uniform() * 50)

** randomly generate income N(10000,100)
**  for each person
gen income = 10000+100*invnormal(uniform())

** GOAL: list top 5 states by income
**       and top 5 states by population
sort state
by state: egen mean_state_income =mean(income)
by state: gen state_pop = _N
by state: keep if _n == 1
gsort -mean_state_income
list state mean_state_income state_pop in 1/5
gsort -state_pop
list state mean_state_income state_pop in 1/5
```

```
. gsort -mean_state_income

. list state mean_state_income state_pop in 1/5

     +------------------------------+
     | state   mean_s~e   state_~p |
     |------------------------------|
  1. |    24   10048.61         18 |
  2. |     7   10045.79         13 |
  3. |    48   10038.06         21 |
  4. |    11   10035.99         22 |
  5. |    39   10034.49         15 |
     +------------------------------+

. gsort -state_pop

. list state mean_state_income state_pop in 1/5

     +------------------------------+
     | state   mean_s~e   state_~p |
     |------------------------------|
  1. |    20   9997.599         33 |
  2. |    47   10024.35         30 |
  3. |    43   10017.19         29 |
  4. |    42   9986.825         29 |
  5. |    13   10027.39         28 |
     +------------------------------+
```

```
** make state population data file (only for 45 states!)
clear
set obs 45
egen state = fill(1 2)
gen  state_population = 1000000*invttail(5,uniform())
save state_populations.dta
list in 1/5

** make state income data file (for all 50 states!)
clear
set obs 1000
gen state  = 1+floor(uniform() * 50)
gen income = 10000 + 100*invnormal(uniform())
sort state
save state_income.dta
list in 1/5

** created merged data set
clear
use state_populations
sort state
save state_populations, replace
clear
use state_income
sort state
merge state using state_populations.dta, uniqusing
tab _merge, missing
tab state if _merge == 2
keep if _merge == 3
drop _merge
```

merge command

NOTE:
_merge==1, obs only in master
_merge==2, obs only in using
_merge==3, obs in both

merge command, con't

```
. ** make state population data file (only for 45 states!)
. Clear
. set obs 45
obs was 0, now 45
. egen state = fill(1 2)
. gen  state_population = 1000000*invttail(5,uniform())
. save state_populations.dta
file state_populations.dta saved
. list in 1/5
     +-------------------+
     | state   state_p~n |
     |-------------------|
  1. |     1    -4682021 |
  2. |     2     1271717 |
  3. |     3   -527176.7 |
  4. |     4    907596.9 |
  5. |     5     1379361 |
     +-------------------+
.
. ** make state income data file (for all 50 states!)
. clear
. set obs 1000
obs was 0, now 1000
. gen state  = 1+floor(uniform() * 50)
. gen income = 10000 + 100*invnormal(uniform())
. sort state
. save state_income.dta
file state_income.dta saved
. list in 1/5
     +------------------+
     | state     income |
     |------------------|
  1. |     1   10056.04 |
  2. |     1   9999.274 |
  3. |     1   10042.95 |
  4. |     1   10095.03 |
  5. |     1   9913.146 |
     +------------------+
```

```
.
. ** created merged data set
. clear
. use  state_populations
. sort state
. save state_populations, replace
file state_populations.dta saved
. clear
. use state_income
. sort state

. merge state using state_populations.dta, uniqusing
variable state does not uniquely identify observations
     in the master data

. tab _merge, missing
     _merge |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |        108       10.80       10.80
          3 |        892       89.20      100.00
------------+-----------------------------------
      Total |      1,000      100.00

. tab state if _merge == 2
no observations
. keep if _merge == 3
(108 observations deleted)
. drop _merge
. save state_merged.dta
file state_merged.dta saved
.
end of do-file
```

reshape command

```
clear
set obs 1000
gen player = 1+floor(uniform() * 100)
bysort player: gen tournament = _n
gen score1 = floor(68 + invnormal(uniform()))
gen score2 = floor(68 + invnormal(uniform()))
gen score3 = floor(68 + invnormal(uniform()))
gen score4 = floor(68 + invnormal(uniform()))

list in 1/3
reshape long score, i(player tournament) j(round)
list in 1/12
```

reshape command, con't

```
.
. list in 1/3

     +-------------------------------------------------------+
     | player   tourna~t   score1   score2   score3   score4 |
     |-------------------------------------------------------|
  1. |      1          1       68       70       68       67 |
  2. |      1          2       68       67       67       67 |
  3. |      1          3       65       67       68       65 |
     +-------------------------------------------------------+

. reshape long score, i(player tournament) j(round)
(note: j = 1 2 3 4)

Data                               wide   ->   long
-----------------------------------------------------------------
Number of obs.                     1000   ->    4000
Number of variables                   6   ->       4
j variable (4 values)                     ->   round
xij variables:
               score1 score2 ... score4   ->   score
-----------------------------------------------------------------

. list in 1/12

     +------------------------------------+
     | player   tourna~t   round   score  |
     |------------------------------------|
  1. |      1          1       1      68  |
  2. |      1          1       2      70  |
  3. |      1          1       3      68  |
  4. |      1          1       4      67  |
  5. |      1          2       1      68  |
     |------------------------------------|
  6. |      1          2       2      67  |
  7. |      1          2       3      67  |
  8. |      1          2       4      67  |
  9. |      1          3       1      65  |
 10. |      1          3       2      67  |
     |------------------------------------|
 11. |      1          3       3      68  |
 12. |      1          3       4      65  |
     +------------------------------------+
```

String functions *(time permitting)*

- Stata has support for basic string operations (length, lowercase, trim, replace).
  - Type “help strfun”

- Here is a small example using regular expressions. This is fairly advanced but can be very helpful sometimes.
  - Here is the data set …

game	price	sectionrow
1	90	FB4,r5
1	75	FB4-5
1	90	4-5
1	80	5FB12
2	80	Field Box 4,12
2	60	4FieldBox12
2	50	Field Box 12, Row 17
2	90	Field Box 2, Rw 5

  - GOAL: Get section number and row number for each observation

Regular expressions

```
clear
insheet using regex.txt
replace sectionrow = subinstr(sectionrow," ", "", .)
local regex = "^([a-zA-Z,.-]*)([0-9]+)([a-zA-Z,.-]*)([0-9]+)$"
gen section = regexs(2) if regexm(sectionrow, "`regex'")
gen row     = regexs(4) if regexm(sectionrow, "`regex'")
list
```

```
. list

     +---------------------------------------------------+
     | game   price          sectionrow   section    row |
     |---------------------------------------------------|
  1. |    1      90              FB4,r5         4      5 |
  2. |    1      75               FB4-5         4      5 |
  3. |    1      90                4-12         4     12 |
  4. |    1      80               5FB12         5     12 |
  5. |    2      80        FieldBox4,12         4     12 |
     |---------------------------------------------------|
  6. |    2      60         4FieldBox12         4     12 |
  7. |    2      50   FieldBox12,Row17        12     17 |
  8. |    2      90      FieldBox2,Rw5          2      5 |
     +---------------------------------------------------+
```

Back to the “standard RA project”

Recall the steps …

1. Read in data
2. Effectively summarize/tabulate data, present graphs
3. Prepare data set for analysis (generate, reshape, parse, encode, recode)
4. Preliminary regressions and output results

To do (4) we will go through a motivating example …

QUESTION: What is the effect of winning the coin toss on the probability of winning a cricket match?

Data

team1	team2	toss	choice	outcome	result	date_str
West Indies	A Priestley's XI	A Priestley's XI	decided to bat	West Indies	by 3 wickets	1897_2_15
Trinidad	A Priestley's XI	Trinidad	decided to bat	Trinidad	by 10 wickets	1897_2_19
Trinidad	A Priestley's XI	Trinidad	decided to bat	Trinidad	by 8 wickets	1897_2_25
Barbados	A Priestley's XI	A Priestley's XI	decided to bat	Barbados	by an innings and 4	1897_1_13
Barbados	A Priestley's XI	A Priestley's XI	decided to field	A Priestley's XI	by 3 wickets	1897_1_18
Barbados	A Priestley's XI	A Priestley's XI	decided to field	Barbados	by 136 runs	1897_1_21
Jamaica	A Priestley's XI	Jamaica	decided to bat	A Priestley's XI	by an innings and 3	1897_3_13
Jamaica	A Priestley's XI	A Priestley's XI	decided to bat	A Priestley's XI	by 10 wickets	1897_3_16
Jamaica	A Priestley's XI	A Priestley's XI	decided to bat	A Priestley's XI	by an innings and 1	1897_3_27
New South Wales	A Shaw's XI	A Shaw's XI	decided to field	A Shaw's XI	by 9 wickets	1886_12_10
New South Wales	A Shaw's XI	A Shaw's XI	decided to field	New South Wales	by 122 runs	1887_2_18
New South Wales	A Shaw's XI	A Shaw's XI	decided to bat	A Shaw's XI	by an innings and 3	1885_1_24
Victoria	A Shaw's XI	Victoria	decided to bat	A Shaw's XI	by 9 wickets	1887_3_4
Victoria	A Shaw's XI	A Shaw's XI	decided to bat	A Shaw's XI	by 118 runs	1884_11_14
New South Wales	A Shaw's XI	A Shaw's XI	decided to bat	New South Wales	by 6 wickets	1886_11_19
New South Wales	A Shaw's XI	New South Wales	decided to bat	A Shaw's XI	by 4 wickets	1884_11_21
Victoria	A Shrewsbury's XI	Victoria	decided to bat	A Shrewsbury's XI	by an innings and 4	1887_12_16
New South Wales	A Shrewsbury's XI	New South Wales	decided to bat	A Shrewsbury's XI	by 10 wickets	1887_12_9
New South Wales	A Shrewsbury's XI	New South Wales	decided to bat	New South Wales	by 153 runs	1888_1_13
New South Wales	A Shrewsbury's XI	New South Wales	decided to field	New South Wales	by 10 wickets	1887_11_10
Trinidad	AB St Hill's XII	Trinidad	decided to bat	Trinidad	by 146 runs	1901_1_10
Victoria	AC MacLaren's XI	Victoria	decided to bat	AC MacLaren's XI	by 8 wickets	1902_2_22
New South Wales	AC MacLaren's XI	New South Wales	decided to bat	AC MacLaren's XI	by an innings and 1	1902_2_31
South Australia	AC MacLaren's XI	South Australia	decided to bat	AC MacLaren's XI	by 6 wickets	1902_3_14

Basic regressions and tables

```
set mem 500m
insheet using cricket14170.txt, tab names

gen year      = real(substr(date_str, 1, 4))
assert(year>1880 & year<2007 & floor(year)==year)
gen won_toss  = (toss    == team1)
gen won_match = (outcome == team1)
summarize won_toss won_match
encode team1, gen(team_id)

regress won_match won_toss, robust
estimates store baseline
xtreg won_match won_toss year, i(team_id) robust
estimates store teamyearFE

estimates table baseline teamyearFE, se stats(r2 N)
```

Basic regressions and tables

```
.
. regress won_match won_toss, robust

Linear regression                                      Number of obs =   42854
                                                       F(  1, 42852) =   51.13
                                                       Prob > F      =  0.0000
                                                       R-squared     =  0.0012
                                                       Root MSE      =   .4968

------------------------------------------------------------------------------
             |               Robust
   won_match |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
    won_toss |    .034355   .0048047     7.15   0.000     .0249377    .0437724
       _cons |   .5361045   .0034752   154.26   0.000      .529293     .542916
------------------------------------------------------------------------------

. estimates store baseline

. xtreg won_match won_toss t, i(team_id) robust fe

Fixed-effects (within) regression               Number of obs      =     42854
Group variable (i): team_id                     Number of groups   =       202

R-sq:  within  = 0.0017                         Obs per group: min =        31
       between = 0.0165                                        avg =     212.1
       overall = 0.0018                                        max =      1336

                                                F(2,42650)         =     35.68
corr(u_i, Xb)  = -0.0002                        Prob > F           =    0.0000

------------------------------------------------------------------------------
             |               Robust
   won_match |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
    won_toss |   .0309134   .0046966     6.58   0.000      .021708    .0401187
           t |    .000444   .0000854     5.20   0.000     .0002767    .0006113
       _cons |   .5378923   .0034018   158.12   0.000     .5312248    .5445598
-------------+----------------------------------------------------------------
     sigma_u |  .15179461
     sigma_e |  .48371236
         rho |  .08964933   (fraction of variance due to u_i)
------------------------------------------------------------------------------

. estimates store teamyearFE

.
. estimates table baseline teamyearFE, se stats(r2 N)

----------------------------------------
    Variable |  baseline    teamyearFE
-------------+--------------------------
    won_toss |  .03435505    .03091337
             |  .00480471    .00469655
           t |               .00044401
             |               .00008536
       _cons |   .5361045    .53789232
             |  .00347524    .00340176
-------------+--------------------------
          r2 |  .00119236    .00168724
           N |      42854        42854
----------------------------------------
                            legend: b/se
```

Global and local variables

- Scalar variables in Stata can be either local or global. Only difference is that global variables are visible outside the current DO file

- Syntax:

```
clear
local  local_variable1  = "local variable"
local  local_variable2  = 14170
global global_variable1 = "global variable"
global global_variable2 = 14170
local  whoa_dude = "$global_variable2"

display "`local_variable1'"
display $global_variable2
display `whoa_dude'
```

```
. di "`local_variable1'"
local variable

. di $global_variable2
14170

. di `whoa_dude'
```

???

Global and local variables, con't

```
local  var1 = "var3"
global var2 = "var3"
local  var3 = 14170

di "`var1'"
di "$var2"
```

`di "``var1''"`
`di "`$var2'"`

NOTE: Last two lines use syntax that is somewhat common in ADO files written by Stata Corp.

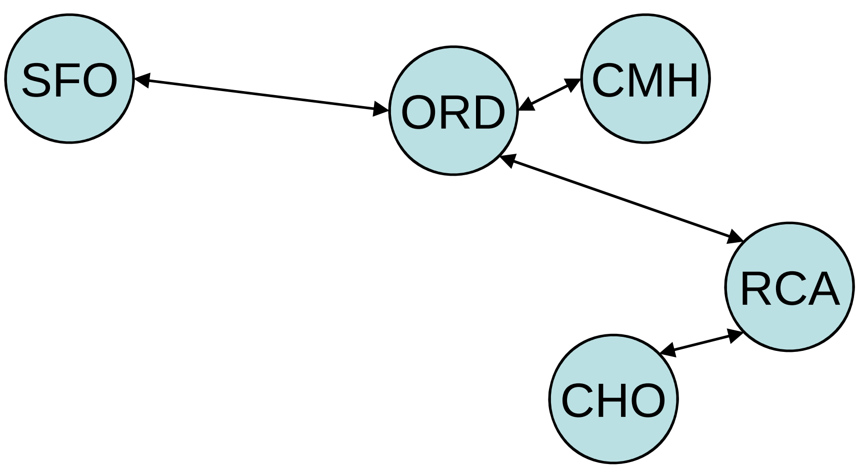
```
. local  var1 = "var3"

. global var2 = "var3"

. local  var3 = 14170

.
. di  "`var1'"
var3

. di  "$var2"
var3

.
. di  "``var1''"
14170

. di  "`$var2'"
14170
```

Control structures (loops)

```
foreach var of varlist math reading writing history science {
   regress `var’ class_size, robust
   estimates store `var’
}
est table `var’

forvalues year = 1940(10)2000 {
   regress log_wage is_female is_black educ_yrs exper_yrs ///
     if year == `year’, robust
   estimates store mincer`year’
}


local i = 1
while (`i’ < 100) {
  display `i’
  local i = `i’ + 1
}

forvalues i = 1/100 {
  display `i’
}
```

Using control structures for data preparation

EXAMPLE: Find all 1-city layover flights given data set of available flights



origin	**dest**	**carrier**
SFO	ORD	Delta
ORD	SFO	Delta
ORD	CMH	Delta
CMH	ORD	Delta
ORD	RCA	Delta
RCA	ORD	Delta
CHO	RCA	Delta
RCA	CHO	Delta

Using control structures for data preparation, con't

<u>LAYOVER BUILDER ALGORITHM</u>

In the raw data, observations are (O, D, C, . , .) tuple where

O = origin

D = destination

C = carrier string

and last two arguments are missing (but will be the second carrier and layover city)

FOR each observation *i* from 1 to N
  FOR each observation *j* from *i*+1 to N
    IF D[i] == O[j] & O[i] != D[j]
      CREATE new tuple (O[i], D[j], C[i], C[j], D[i])

Control structures for Data Preparation

```
insheet using airlines.txt, tab names
gen carrier2 = ""
gen layover = ""
local numobs = _N
forvalues i = 1/`numobs' {
  di "doing observations `i' ..."
  forvalues j = 1/`numobs' {
    if (dest[`i'] == origin[`j'] & origin[`i'] != dest[`j']) {
       **create new observation for layover flight
       local newobs = _N + 1
       set obs `newobs'
       quietly {
        replace origin   = origin[`i']  if _n == `newobs'
        replace dest     = dest[`j']    if _n == `newobs'
        replace carrier  = carrier[`i'] if _n == `newobs'
        replace carrier2 = carrier[`j'] if _n == `newobs'
        replace layover  = dest[`i']    if _n == `newobs'
       }
    }
  }
}
```

Control structures for Data Preparation

```
.
. list

     +------------------------------------------------------+
     | origin   dest   carrier   carrier2   layover |
     |------------------------------------------------------|
  1. |    SFO    ORD     Delta                      |
  2. |    ORD    SFO     Delta                      |
  3. |    ORD    CMH     Delta                      |
  4. |    CMH    ORD     Delta                      |
  5. |    ORD    RCA     Delta                      |
     |------------------------------------------------------|
  6. |    RCA    ORD     Delta                      |
  7. |    CHO    RCA     Delta                      |
  8. |    RCA    CHO     Delta                      |
  9. |    SFO    CMH     Delta      Delta       ORD |
 10. |    SFO    RCA     Delta      Delta       ORD |
     |------------------------------------------------------|
 11. |    CMH    SFO     Delta      Delta       ORD |
 12. |    CMH    RCA     Delta      Delta       ORD |
 13. |    ORD    CHO     Delta      Delta       RCA |
 14. |    RCA    SFO     Delta      Delta       ORD |
 15. |    RCA    CMH     Delta      Delta       ORD |
     |------------------------------------------------------|
 16. |    CHO    ORD     Delta      Delta       RCA |
     +------------------------------------------------------+
```

A diversion: Intro to Algorithms

- The runtime of this algorithm (as written) is $O(N^2)$ time, where N is the number of observations in the data set (*)

- Whether using Matlab or C, the runtime will be asymptotically equivalent. This is important to keep in mind when thinking about making calls to C. Most of the time you only get a proportional increase speed, not an asymptotic increase. In some cases, thinking harder about getting your algorithm to run in $O(N*\log(N))$ time instead of $O(N^2)$ is much more important than making calls to a programming language with less overhead.

- In order to improve asymptotic runtime efficiency, need better **data structures** than just arrays and/or matrices – specifically, need hash tables (**)

- Perl, Java, and C++ all have built-in hash tables. Some C implementations also available. We will cover this in more detail in Perl class this afternoon.

- With proper functioning hash tables, the runtime in previous algorithm can be reduced to $O(N)$ (!!)

(*) the runtime is actually $O(N^3)$ in Stata but would be $O(N^2)$ in a standard Matlab and/or C implementation for reasons we will discuss in the next lecture

(**) also called “Associative Arrays”, “Lookup Tables” or “Index Files”

Exercises

- Go to following URL:
  http://web.mit.edu/econ-gea/14.170/exercises/

- Download each DO file
  - No DTA files! All data files are loaded from the web (see "help **webuse**")
  - First run DO file "AS IS" before doing anything else (this will save data files in your local directory)

- 3 exercises (in increasing difficulty and – in my opinion – decreasing importance)

  A: Preparing a data set, running some preliminary regressions, and outputting results

  B: More with finding layover flights

  C: Using regular expressions to parse data