Applied Economics I

David Strömberg, Department of Economics, SU

Lecture 2: Programming principles and Stata

Search in Stata

Command	what	where
help *command name*	exact command name	official stata
search *keywords*	keywords	official stata
findit *keywords*	keywords	official stata + online

Some useful Stata commands

- merge
  - Combines data set horizontally based on keys.
  - merge m:1 state using state

county_pop.dta

county	state	population
36037	NY	3817735
36038	NY	422999
36039	NY	324920
36040	NY	143432
37001	VA	3228290
37002	VA	449499
37003	VA	383888
37004	VA	483829

state	population	region
NY	43320903	1
VA	7173000	3

	county	state	population	region	_merge
1	36037	NY	3817735	1	matched (3)
2	36038	NY	422999	1	matched (3)
3	36039	NY	324920	1	matched (3)
4	36040	NY	143432	1	matched (3)
5	37001	VA	3228290	3	matched (3)
6	37002	VA	449499	3	matched (3)
7	37003	VA	383888	3	matched (3)
8	37004	VA	483829	3	matched (3)

merge

- Creates variable _merge with values:
  1. master only
  2. using only
  3. match
  4. match update
  5. match conflict

- Some useful options
  - keep(master match)
  - keepusing(varlist)
  - update
  - sorted
  - nogen

reshape

- reshape long faminc, i(famid) j(year)
- reshape wide faminc, , i(famid) j(year)

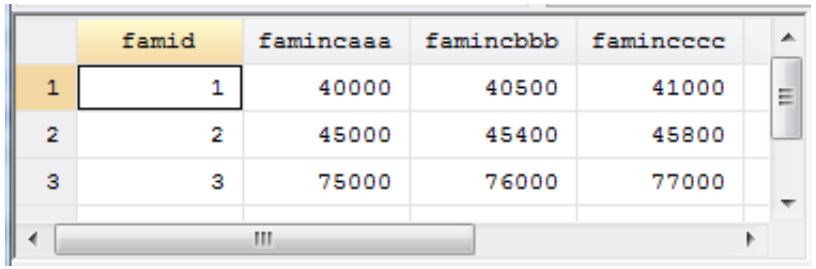
stub *observation* *value*

	famid	faminc96	faminc97	faminc98
1	3	75000	76000	77000
2	1	40000	40500	41000
3	2	45000	45400	45800

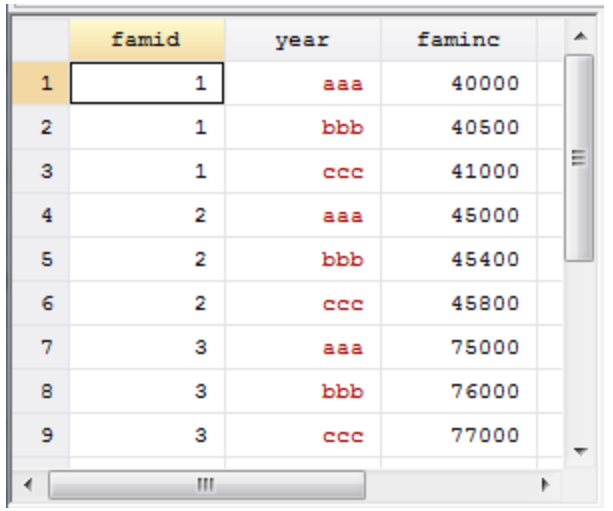
	famid	year	faminc
1	1	96	40000
2	1	97	40500
3	1	98	41000
4	2	96	45000
5	2	97	45400
6	2	98	45800
7	3	96	75000
8	3	97	76000
9	3	98	77000

reshape using strings

- reshape long faminc, i(famid) j(year) string
- reshape wide faminc, , i(famid) j(year) string



	famid	famincaaa	famincbbb	famincccc
1	1	40000	40500	41000
2	2	45000	45400	45800
3	3	75000	76000	77000



	famid	year	faminc
1	1	aaa	40000
2	1	bbb	40500
3	1	ccc	41000
4	2	aaa	45000
5	2	bbb	45400
6	2	ccc	45800
7	3	aaa	75000
8	3	bbb	76000
9	3	ccc	77000

String to numeric conversion

- tostring x, replace force
- destring x, gen(x2) force
- encode gender, gen(gender_int)
- decode gender_int, gen(gender2)

	x	x2	gender	gender_int	gender2
1	.0181898233	.01818982	female	female	female
2	.8708924055	.87089241	male	male	male
3	.7101488709	.71014887	male	male	male
4	.2058176845	.20581768	female	female	female
5	.1334635764	.13346358	female	female	female
6	.2760108411	.27601084	female	female	female
7	.6522316933	.65223169	male	male	male

```
Contains data
  obs:           100
 vars:             2
 size:         1,000
-------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label
-------------------------------------------------------------------------------
x               float   %9.0g
gender          str6    %9s
-------------------------------------------------------------------------------
```

dummy and interaction variables

- xi: regress LWKLYWGE i.QOB

- xi i.QOB, prefix(QOB)

	QOB	QOBQOB_2	QOBQOB_3	QOBQOB_4
1	3	0	1	0
2	1	0	0	0
3	3	0	1	0
4	4	0	0	1

- By default, i.varname omits the dummy corresponding to the smallest value of varname.
- char _dta[omit] prevalent : drop most prevalent dummy

LWKLYWGE	Coef.	Std. Err.
_IQOB_2	.0042928	.0037065
_IQOB_3	.0130902	.0036593
_IQOB_4	.0093453	.0037269
_cons	5.148471	.0026024

- char QOB[omit] 3 : drop specific dummy

LWKLYWGE	Coef.	Std. Err.
_IQOB_1	-.0042928	.0037065
_IQOB_3	.0087974	.0036857
_IQOB_4	.0050526	.0037528
_cons	5.152764	.0026393

Macros

- Use as shorthand – you type a macro name but are actually referring to some numerical value or a string of characters.
  - Local macros only work within the program or do-file in which they are created.
  - Global macros work within a Stata session. Avoid.

```
. local ctyname Sweden

. di "`ctyname'"
Sweden
```

Useful macro facts

- = means evaluate before assignment
- referenced by `’
  - `’ evaluates what’s inside brackets.

```
. local s 2+2

. di "`s'"
2+2

. di `s'
4
```

```
. local s=2+2

. di "`s'"
4
```

```
. di `=2+2'
4

. di `"=2+2"'
=2+2

. di "`=2+2'"
4
```

```
. local ctyname Sweden

. di "`ctyname'"
Sweden

. di `ctyname'
Sweden not found
```

- Self-references allowed

```
. local controls "RACE MARRIED SMSA NEWENG MIDATL ENOCENT WNOCENT SOATL ESOCENT WSOCENT MT"

. local controls "`controls' YR20-YR28"

. di "`controls'"
RACE MARRIED SMSA NEWENG MIDATL ENOCENT WNOCENT SOATL ESOCENT WSOCENT MT YR20-YR28
```

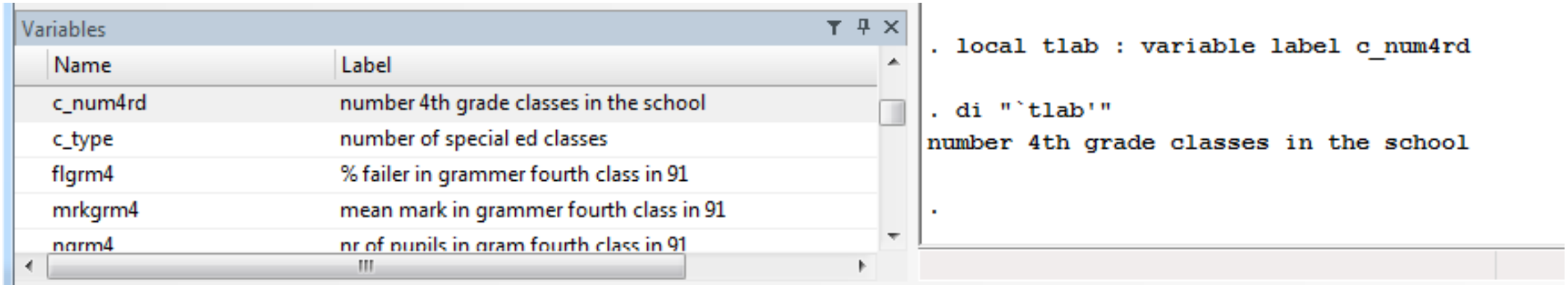
Macro extended functions

- local mname : extended macro function
  - colon is equivalent to = for regular macro

- Examples
  - local lbl : variable label myvar
    - the variable label associated with variable myvar will be stored in macro lbl
  - local filenames : dir "." files "*.dta“
    - macro filenames will contain the names of all the .dta datasets in the current directory
  - local xi : word `i' of `list‘
    xi will contain the `i'th word (element) of `list

Macro extended functions

- Extracting data attributes: example variable label



Macro extended functions useful for loops

- Extracting file names or file paths, to loop over files.

```
. local list : dir . files "*.do"

. foreach file in `list' {
  2.          di "`file'"
  3.          }
master.do
tableiv.do
tableiv_edit.do
tablev.do
tablevi.do
```

```
. local dirlist : dir . dirs "*"

. foreach dir in `dirlist' {
  2.          di "`dir'"
  3.          }
.git
Analysis
Build
Other
Raw
```

Other useful extended macro functions

- Other useful extended macro functions
  - local y : word 1 of `rvarlist‘
  - local p : word count `xS'
- Macro extended function for manipulating lists
  - A list is a space-separated set of elements listed one after the other.
  - Syntax
    - local *macroname* : list *function*
      - local x : list rvarlist – y
      - local newvarlist : list x | y
- c()
  - local curdir "`c(pwd)‘”

```
. local rvarlist "LWKLYWGE EDUC"

. local y : word 1 of `rvarlist'

. di "`y'"
LWKLYWGE

. local x : list rvarlist - y

. di "`x'"
EDUC

. local p : word count `x'

. di `p'
1

. local newvarlist : list x | y

. di "`newvarlist'"
EDUC LWKLYWGE
```

Extracting variable names and to loop over variables.

- ds -- List variables matching name patterns or other characteristics
- For example, to verify variables
  - summarize numeric variables
  - tab values for integer variables
  - check # unique values for string vars.

```
. ds, has(type int)
c_size    c_girls   nmth4     nmath_n
c_boys    ngrm4     cohsize   nverb_n

. foreach var in `r(varlist)' {
  2.          di "`var'"
  3.          }
c_size
c_boys
c_girls
ngrm4
nmth4
cohsize
nmath_n
nverb_n

. ds, has(type string)
c_tip     townname

. ds, has(type numeric)
schlcode  flgrm4    classid   passverb  verb_n
c_size    mrkgrm4   classize  studchk   flverb_n
c_boys    ngrm4     _type_    misskov2  nverb_n
c_girls   flmth4    _freq_    missagg   impute
c_numcl   mrkmth4   cohsize   nmiss_k   nverb_m
```

Loops: foreach, forvalues, while

- foreach lname
  - in any_list
  - of varlist
  - of numlist
  - etc
  - {
  - …
  - }

```
. local flist : dir "Build\Code" files "*"

. foreach file in `flist' {
  2.          di "`file'"
  3.          }
TableIV_data.do
TableVI_data.do
TableV_data.do



foreach var of varlist `varlist' {
        ...
}

foreach num of numlist 1 4/8 13(2)21 103 {
        display `num'
}
```

Loops: forvalues

- forvalues name = range {
  - …
  - }

- Properties
  - loop over consecutive values
  - faster than foreach
  - handles larger lists (does not store numlist before execution)

```
. forvalues i = 1(1)100 {
  2.        generate x`i' = runiform()
  3. }
```

```
. forvalues i=1(1)10000 {
  2. }

. foreach num of numlist 1(1)10000 {
  2. }
invalid numlist has too many elements
```

Loops: while

- while condition {
  - …
  - }

- Increment and decrement operators

  ++i increment before

  --i decrement before

  i++ increment after

  i-- decrement after

```
. local i=1

. while `i'<=2 {
  2.          di "`i'"
  3.          local i = `i' + 1
  4.          }
1
2

. local i=1

. while `++i'<=2 {
  2.          di "`i'"
  3.          }
2

. local i=1

. while `i++'<=2 {
  2.          di "`i'"
  3.          }
2
3
```

String functions in Stata

- Stata has some simple string functions.
  - help string functions
- Functions using regular expressions
  - allows for matching complex patterns of text with minimal effort
  - commands
    - regexm(s,re) : performs match
    - regexr(s1,re,s2): replaces the first substring within s1 that matches re with s2 and returns the resulting string
    - regexs(n): subexpression n from a previous regexm()

Regular expressions

- Regular expressions or regexps originated in 1956, when mathematician Stephen Cole Kleene described regular languages using his mathematical notation called regular sets.
- Many programming languages provide regexp capabilities, some built-in, for example Perl, JavaScript, Ruby, AWK, and Tcl, and others via a standard library, for example Java and Python.

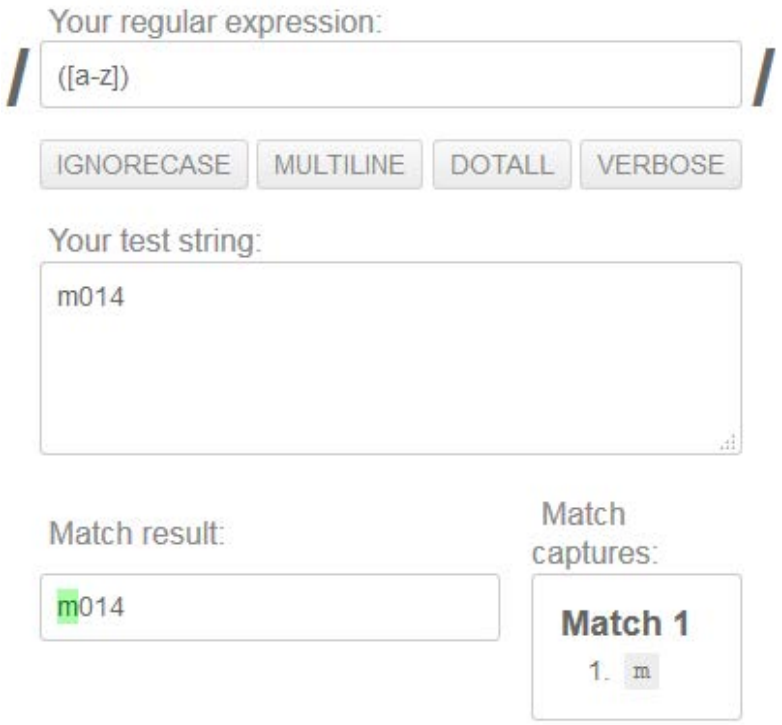
- https://www.stata.com/support/faqs/data-management/regular-expressions/

() Define a match group.

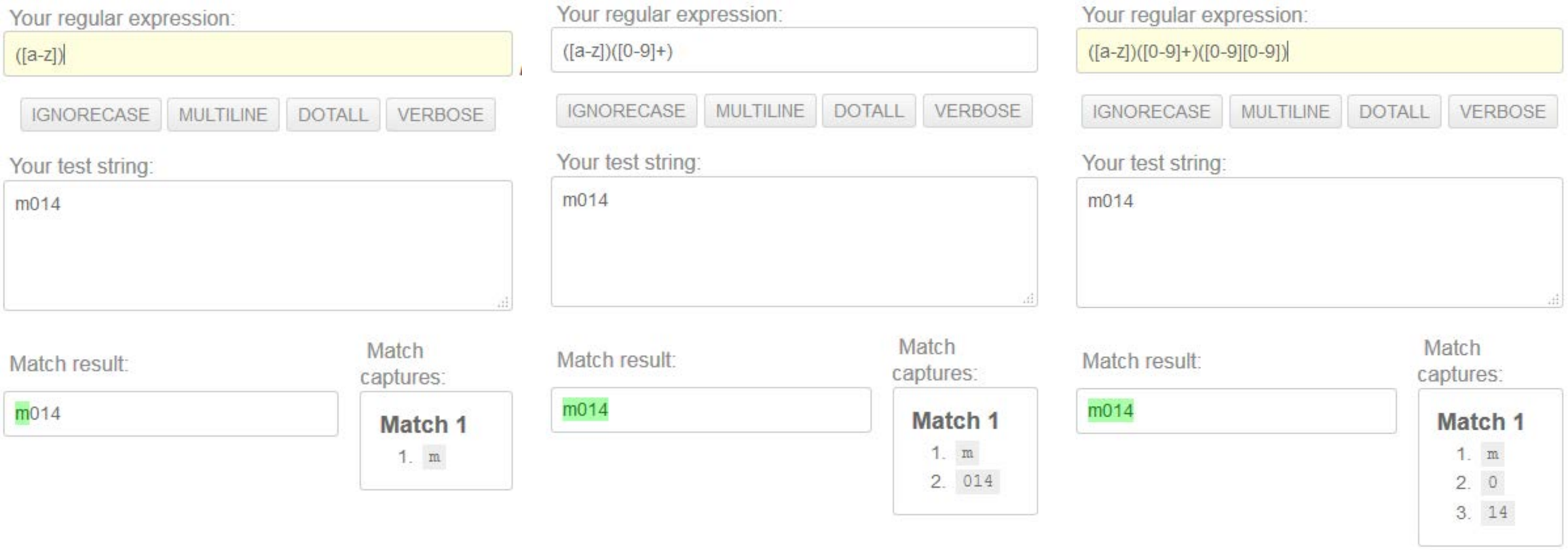
[] Match any of a set of characters.

Counting	
*	Asterisk means “match zero or more” of the preceding expression.
+	Plus sign means “match one or more” of the preceding expression.
?	Question mark means “match either zero or one” of the preceding expression.
Characters	
a–z	The dash operator means “match a range of characters or numbers”. The “a” and “z” are merely an example. It could also be 0–9, 5–8, F–M, etc.
.	Period means “match any character”.
\	A backslash is used as an escape character to match characters that would otherwise be interpreted as a regular-expression operator.
Anchors	
^	When placed at the beginning of a regular expression, the caret means “match expression at beginning of string”. This character can be thought of as an “anchor” character since it does not directly match a character, only the location of the match.
$	When the dollar sign is placed at the end of a regular expression, it means “match expression at end of string”. This is the other anchor character.

- https://pythex.org/



- Check your expression at https://pythex.org/



```
. gen x= regexs(1) if regexm("m014","([a-z])([0-9]+)([0-9][0-9])")
. di regexs(1)
m
. di regexs(2)
0
. di regexs(3)
14
. di regexs(2) + "-" + regexs(3)
0-14
```

```
gen sex= regexs(1)                  if regexm(column,"([a-z])([0-9]+)([0-9][0-9])")
gen age = regexs(2)+"-"+regexs(3)   if regexm(column,"([a-z])([0-9]+)([0-9][0-9])")
```

country	year	column	cases
AD	2000	m014	0
AD	2000	m1524	0
AD	2000	m2534	1
AD	2000	m3544	0
AD	2000	m4554	0
AD	2000	m5564	0
AD	2000	m65	0
AE	2000	m014	2
AE	2000	m1524	4
AE	2000	m2534	4
AE	2000	m3544	6
AE	2000	m4554	5
AE	2000	m5564	12
AE	2000	m65	10
AE	2000	f014	3

(a) Molten data

country	year	sex	age	cases
AD	2000	m	0–14	0
AD	2000	m	15–24	0
AD	2000	m	25–34	1
AD	2000	m	35–44	0
AD	2000	m	45–54	0
AD	2000	m	55–64	0
AD	2000	m	65+	0
AE	2000	m	0–14	2
AE	2000	m	15–24	4
AE	2000	m	25–34	4
AE	2000	m	35–44	6
AE	2000	m	45–54	5
AE	2000	m	55–64	12
AE	2000	m	65+	10
AE	2000	f	0-14	3

(b) Tidy data

Output

- outreg2 to output regression tables (also check estout/estab)
  - net install outreg2 (or findit outreg to get instructions)

```
reg  LWKLYWGE EDUC  YR20-YR28;
outreg2 EDUC using  `outdir'/TableIV,    excel
                                         addtext(9 Year-of-birth dummies,Yes, 8 Region of residence dummies, No)
                                         se alpha(0.01, 0.05, 0.1)
                                         dec(4)
                                         nocons
                                         replace;
```

Source	SS	df	MS		
Model	17934.8419	10	1793.48419	Number of obs	= 247,199
Residual	86918.1779	247,188	.351627821	F(10, 247188)	= 5100.52
				Prob > F	= 0.0000
				R-squared	= 0.1710
				Adj R-squared	= 0.1710
Total	104853.02	247,198	.424166133	Root MSE	= .59298

LWKLYWGE	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
EDUC	.0801595	.0003552	225.67	0.000	.0794633	.0808557
YR20	.023484	.0053878	4.36	0.000	.0129241	.0340439
YR21	.02899	.0053167	5.45	0.000	.0185693	.0394107
YR22	.0232415	.0053556	4.34	0.000	.0127447	.0337383

	A	B
1		
2		(1)
3	VARIABLES	WKLYWGE
4		
5	EDUC	0.0802***
6		(0.0004)
7		
8	Observations	247,199
9	R-squared	0.1710
10	9 Year-of-birth dummies	Yes
11	8 Region of residence dummies	No
12	Standard errors in parentheses	
13	*** p<0.01, ** p<0.05, * p<0.1	

OLS AND TSLS ESTIMATES

Independent variable	(1) OLS
Years of education	0.0802
	(0.0004)
Race (1 = black)	—
SMSA (1 = center city)	—
Married (1 = married)	—
9 Year-of-birth dummies	Yes
8 Region of residence dummies	No
Age	—
Age-squared	—
χ^2 [dof]	—

Output

- outreg2 can also be used to output e.g. summary tables.

```
help outreg2
```

Titles & related stuffs

outreg2 — Arrange regression, summary, and tabulation into an illustrative table
seeout — Opens a tab-delimited table in the data browser
shellout — Opens documents and their programs from inside Stata

logout — Converts log or ASCII files into various output formats
mkest — Convert variables to estimates matrix

Examples

0. Basic game plan
1. Prefix and -bys:-
2. Running loops
3. Stored estimates
4. Shorthand
5. Decimal places
6. Sideway display
7. Summary tables
8. Drop/keep/order variables
9. Adding r( ) e( ) scalars
10. Odds ratios and rrr
11. Marginal Effects
12. Word or Excel files
13. TeX files
14. Adding column titles or notes
15. n-way cross-tabulation
16. Group summary table
17. Fixed effects or legends
18. Stats transformation/manipulation
19. TBA
20. Insert r-class cmd( ) outputs

Different versions of **outreg** and **outreg2**

Output

```
#d;
outreg2  using Analysis/Output/TableIV_sumstat,
    dec(2) excel sum(log) label
    title(Table 3. Summary Statistics)
    keep( LWKLYWGE EDUC `controls') replace;
```

Table 3. Summary Statistics

VARIABLES	(1) N	(2) mean	(3) sd	(4) min	(5) max
EDUC	247,199.00	11.49	3.36	0.00	18.00
ENOCENT	247,199.00	0.22	0.41	0.00	1.00
ESOCENT	247,199.00	0.06	0.23	0.00	1.00
LWKLYWGE	247,199.00	5.16	0.65	-0.02	8.95
MARRIED	247,199.00	0.89	0.31	0.00	1.00
MIDATL	247,199.00	0.20	0.40	0.00	1.00
MT	247,199.00	0.04	0.19	0.00	1.00
NEWENG	247,199.00	0.05	0.22	0.00	1.00
RACE	247,199.00	0.08	0.27	0.00	1.00
SMSA	247,199.00	0.30	0.46	0.00	1.00
SOATL	247,199.00	0.15	0.36	0.00	1.00
WNOCENT	247,199.00	0.07	0.26	0.00	1.00
WSOCENT	247,199.00	0.09	0.28	0.00	1.00

Stata 16: Data frames.
Mutiple datasets in memory

- Datasets in memory stored in named frames.
  - **default**, initial frame.
- You can create frames, and delete them, and rename them.

```
. frame create framename
. frame drop framename
. frame rename oldname newname
```

- Another way of working with frames is

```
. frame framename {
        stata_command
        stata_command
        .
        .
  }
```

```
. frame framename: one_stata_command
```

Example: serial dictator allocation

(Projects\pat\Programs\serial_dictator_allocation.do)

Applicant preferences over choices (programs)

Data Editor (Browse) - choices_small_1971.dta@morpork

File Edit View Data Tools

3R x 6C | 1971

	prodar	sgyreg	rank	applicant	val	choice
1	1971	1	1	58	1	9
2	1971	1	1	58	2	13
3	1971	1	1	58	3	8
4	1971	1	2	1506	1	37
5	1971	1	2	1506	2	38
6	1971	1	2	1506	3	9
7	1971	1	2	1506	4	13
8	1971	1	3	1753	1	37
9	1971	1	3	1753	2	9

Number of slots per choice

Data Editor (Browse) - slots_small_1971.dt

File Edit View Data Tools

1R x 2C | 9

	choice	slot
1	1	38
2	2	32
3	3	8
4	4	0
5	5	117
6	6	85
7	7	0
8	8	1007
9	9	759

Example: serial dictator allocation

(Projects\pat\Programs\serial_dictator_allocation.do)

```
forvalues prodar = 1971/1972 {

    * Load applicant and slot data sets
    clear frames

    use choices_small_`prodar', replace
    sum rank
    local max_rank = r(max)

    * Load the number of slots per choice
    frame create slots
    frame slots: {
        use slots_small_`prodar'
        }

    * Create a frame were the  results are stored.
    frame create allocation applicant choice

    timer clear
    timer on 1
    forvalues i=1/`max_rank' {
        quietly {

        * Select applicant
        frame put if rank==`i', into(applicant)

        * Identify preferred choice among set with remaining slots, and post allocation
        frame applicant: {
            * Merge is m:1 since some apply to same choice in more than one val
            frlink m:1 choice , frame(slots)
            frget slot, from(slots)
            drop if slot<1
            sort applicant val
            by applicant: keep if _n==1
            local N=_N
            forvalues j=1/`N' {
                local applicant = applicant[`j']
                local choice = choice[`j']
                frame post allocation (`applicant') (`choice')
                }
            }
        * Remove taken slots
        frame slots:{
            frlink 1:1 choice , frame(applicant)
            replace slot=slot-1 if applicant!=.
            drop applicant
            }
        frame drop applicant
        }
        }
    timer off 1
    timer list

    frame change allocation
    save allocation_serial_dictator_`prodar', replace
    }
```

Programs

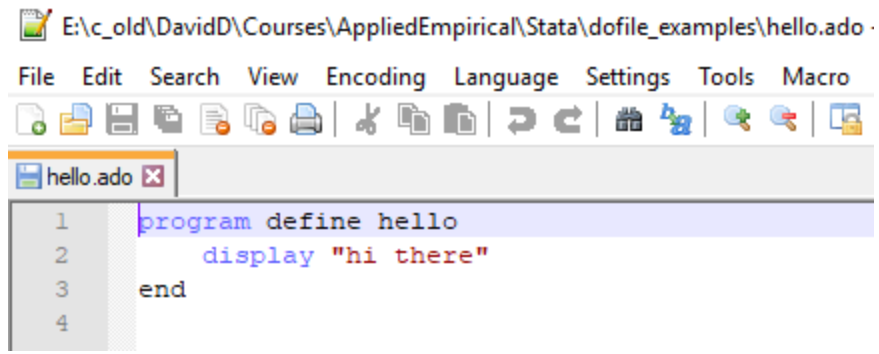
- A program in STATA is a collection of STATA code that starts with the command program define and ends with the command end.
- You execute it by typing the program name.

```
. program define hello
  1.          display "hi there"
  2. end

.
. hello
hi there
```

Programs

- A program in is defined
  1. in a do-file prior to calling it, or
  2. in a ado-file in a adopath folder.

```
. hello
command hello is unrecognized
r(199);

. program define hello
  1.
. display "hi there"
  2.
. end

. hello
hi there
```




```
. hello
command hello is unrecognized
r(199);

. adopath + e:\c_old\DavidD\Courses\AppliedEmpirical\Stata\dofile_examples\

. hello
hi there
```

Programs

- To redefine the program you must first drop it.

- List names of programs stored in memory
  - program dir { pgmname [pgmname [...]] | _all | _allado }
- Eliminate program from memory
  - program drop { pgmname [pgmname [...]] | _all | _allado }
- List content of program
  - program list [pgmname [pgmname [...]] | _all]

```
. program drop hello

. program define hello
  1.          display "hi again"
  2. end

.
. hello
hi again
```

Program positional arguments

- Program arguments are passed to programs via local macros: `1’, `2’,….

```
. program define listargs
  1.         display "The 1st argument you typed is: `1'"
  2.         display "The 2nd argument you typed is: `2'"
  3.         display "The 3rd argument you typed is: `3'"
  4.         display "The 4th argument you typed is: `4'"
  5. end

.
. listargs this is a test
The 1st argument you typed is: this
The 2nd argument you typed is: is
The 3rd argument you typed is: a
The 4th argument you typed is: test

. listargs "this is" "a test"
The 1st argument you typed is: this is
The 2nd argument you typed is: a test
The 3rd argument you typed is:
The 4th argument you typed is:
```

Program named arguments: args

- Specify a list of local macros after the command args in the program.
- When calling the program, pass the values of the locals in the same order.

```
. program define listargs_args
  1. args first second third fourth
  2. display "The 1st argument you typed is: `first'"
  3. display "The 2nd argument you typed is: `second'"
  4. display "The 3rd argument you typed is: `third'"
  5. display "The 4th argument you typed is: `fourth'"
  6. end


. listargs_args hello world here i am
The 1st argument you typed is: hello
The 2nd argument you typed is: world
The 3rd argument you typed is: here
The 4th argument you typed is: i
```

Program named arguments

- You obtain better-named positional arguments by using the args command.

```
program progname
        args argnames
        ...
end
```

- This is better because meaningful names produce cleaner code than `1', `2'.

Program to generate data set
with n observations for x in range a to b.

```
program rng                                    // argu
        clear
        set obs '1'
        generate x = (_n-1)/(_N-1)*('3'-'2')+'2'
end
```

```
program rng
        args n a b
        clear
        set obs 'n'
        generate x = (_n-1)/(_N-1)*('b'-'a')+'a'
end
```

Program syntax

- If your command uses standard Stata syntax with arguments being:
  - a list of variables, possibly a weight, if or in clause, a bunch of options,
  - then you can use Stata's own parser, which stores all these elements in local macros.

```
program leaveout_mean
    syntax, invar(varname) outvar(name) byvar(varname)
    tempvar tot_invar count_invar
    egen `tot_invar'= total(`invar'), by(`byvar')
    egen `count_invar'= count(`invar'), by(`byvar')
    gen `outvar' = (`tot_invar' - `invar') / (`count_invar' - 1)
end

leaveout_mean, invar(pc_potato) outvar(leaveout_state_pc_potato) byvar(state)
leaveout_mean, invar(pc_potato) outvar(leaveout_metro_pc_potato) byvar(metro)
leaveout_mean, invar(hh_potato) outvar(leaveout_metro_hh_potato) byvar(metro)
```

cmd [*varlist* | *namelist* | *anything*]
[*if*]
[*in*]
[using *filename*]
[= *exp*]
[*weight*]
[, *options*]

Program, rclass

```
. ******* PARAMETRIC BOOTSTRAP
.
. * First estimate the model to get the parameters for the bootstrap
. use bootdata.dta, replace

. reg docvis chronic
```

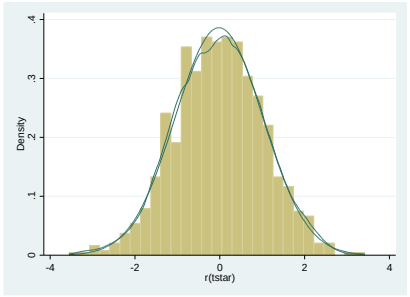
Source	SS	df	MS	Number of obs	=	50
				F(1, 48)	=	3.84
Model	222.141111	1	222.141111	Prob > F	=	0.0558
Residual	2775.13889	48	57.8153935	R-squared	=	0.0741
				Adj R-squared	=	0.0548
Total	2997.28	49	61.1689796	Root MSE	=	7.6036

docvis	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
chronic	4.694444	2.394923	1.96	0.056	-.1208701	9.509759
_cons	2.805556	1.267274	2.21	0.032	.2575306	5.353581

```
. local alpha = _b[_cons]

. local theta = _b[chronic]

. local setheta = _se[chronic]

. local rmse = e(rmse)
```

```
* Program to simulate y using model and parameters
capture program drop bootparametric
program bootparametric, rclass
  version 10.1
  args alpha theta rmse
  capture drop docvis_i
  generate docvis_i = `alpha' + `theta' * chronic + `rmse'*rnormal()
  quietly reg docvis_i chronic
  return scalar tstar = (_b[chronic]-`theta')/_se[chronic]
end
```

```
.
. * Check the program by running once
. set seed 10101

. bootparametric `alpha' `theta' `rmse'

. return list

scalars:
              r(tstar) =  -.7238617914576307

.
. * Parametric bootstrap for the parameters
. simulate tstar=r(tstar), seed(10101) reps(999) nodots  ///
>   saving(percentilet3, replace): bootparametric  `alpha' `theta' `rmse'

      command:  bootparametric 2.805555555555555 4.694444444444445 7.603643437097673
        tstar:  r(tstar)


.
. histogram tstar, normal kdensity
(bin=29, start=-3.5523593, width=.24049872)
```



Passing arguments to dofiles in batch mode

- This can also be used to start parallel instances of Stata from batch mode.
- The code below starts 10 parallel instances of stata to count words in a Linux environment.

```
set rmsg on
! (stata -b do "wordcount0" &); (stata -b do "wordcount1" &)
! (stata -b do "wordcount2" &); (stata -b do "wordcount3" &)
! (stata -b do "wordcount4" &); (stata -b do "wordcount5" &)
! (stata -b do "wordcount6" &); (stata -b do "wordcount7" &)
! (stata -b do "wordcount8" &); (stata -b do "wordcount10pct" &)
```

Passing arguments to other programs

- You can similarly pass arguments to other programs using Stata's shell command

```
! perl `perlpath'/wordcount.pl "`path'" "`in'" "`out'"
! perl `perlpath'/wordpostcount.pl "`path'" "`in'" "`out2'"
```

- You can similarly execute Windows commpand prompt commands: copy, mkdir, etc, from Stata using the shell command "!" https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/windows-commands

ado-files

- ado-files, or ”automatic do-files”
  - filename ends with .ado
  - stored in ado directory
- When you type a command, Stata checks the ado directories to see if there is an ado file with that name. If there is, Stata automatically runs the ado file that defines the program and then executes it. I.e. like using a built-in Stata command.
- set adopath using ”adopath + path”

```
. adopath + "e:\c_old\DavidD\Courses\AppliedEmpirical\Stata\dofile_examples\"
  [1]  (BASE)      "C:\Program Files (x86)\Stata14\ado\base/"
  [2]  (SITE)      "C:\Program Files (x86)\Stata14\ado\site/"
  [3]              "."
  [4]  (PERSONAL)  "c:\ado\personal/"
  [5]  (PLUS)      "c:\ado\plus/"
  [6]  (OLDPLACE)  "c:\ado/"
  [7]              "e:\c_old\DavidD\Courses\AppliedEmpirical\Stata\dofile_examples\"
```

ado-files

listargs_args.ado | examples.do

```
1  program define listargs_args
2      args first second third fourth
3
4      display "The 1st argument you typed is: `first'"
5      display "The 2nd argument you typed is: `second'"
6      display "The 3rd argument you typed is: `third'"
7      display "The 4th argument you typed is: `fourth'"
8  end
9
```

```
. listargs_args this is a test
The 1st argument you typed is: this
The 2nd argument you typed is: is
The 3rd argument you typed is: a
The 4th argument you typed is: test
```

Why use programs?

- Eliminate duplication: don’t write the same code twice.
- Modularize: split into programs and functions.
- Write clean code, use names efficiently.

```
egen total_pc_potato = total(pc_potato), by(state)
egen total_obs = count(pc_potato), by(state)
gen leaveout_state_pc_potato = (total_pc_potato - pc_potato) / (total_obs - 1)

egen total_pc_potato = total(pc_potato), by(metroarea)
egen total_obs = count(pc_potato), by(state)
gen leaveout_metro_pc_potato = (total_pc_potato - pc_potato) / (total_obs - 1)

egen total_hh_potato = total(hh_potato), by(metroarea)
egen total_obs = count(hh_potato), by(state)
gen leaveout_metro_hh_potato = (total_hh_potato - pc_potato) / (total_obs - 1)
```

```
program leaveout_mean
    syntax, invar(varname) outvar(name) byvar(varname)
    tempvar tot_invar count_invar
    egen `tot_invar'= total(`invar'), by(`byvar')
    egen `count_invar'= count(`invar'), by(`byvar')
    gen `outvar' = (`tot_invar' - `invar') / (`count_invar' - 1)
end

leaveout_mean, invar(pc_potato) outvar(leaveout_state_pc_potato) byvar(state)
leaveout_mean, invar(pc_potato) outvar(leaveout_metro_pc_potato) byvar(metro)
leaveout_mean, invar(hh_potato) outvar(leaveout_metro_hh_potato) byvar(metro)
```

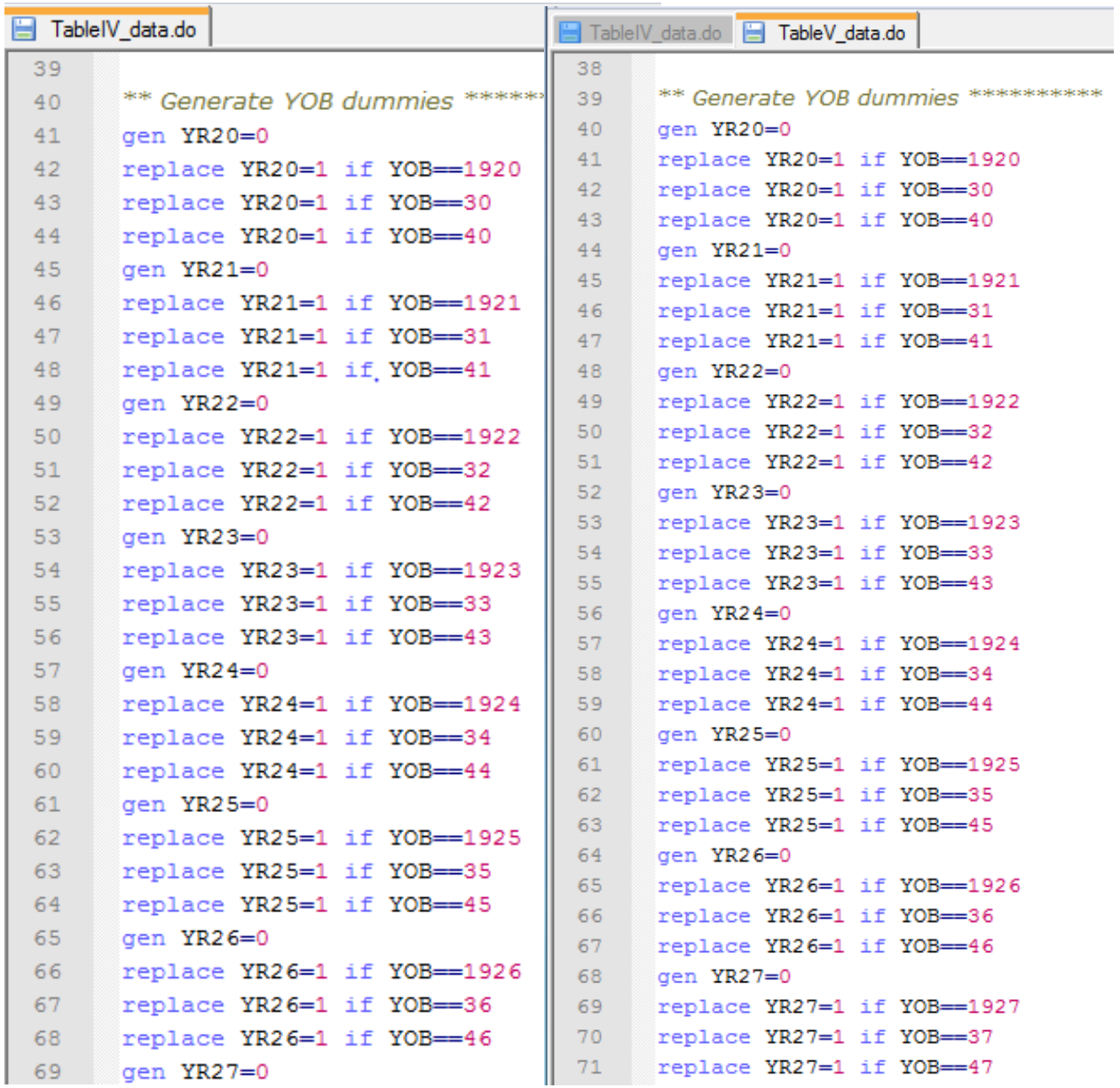
Programming Practice

Good practice

- Eliminate duplication: don’t write the same code twice.
- Modularize: split into programs and functions.
- Write clean code, code should be self-documenting:
  - use names efficiently
  - comment
  - make dependencies explicit
  - programming is communication (often with your future self).
- Test programs using a simple example or test data set.

Eliminate duplication

- Duplication
  - Makes code intransparent – easy to make mistakes.
  - Same code in different places – easy to forget to update everywhere.
- Use
  - Loops
  - Functions
  - Macros

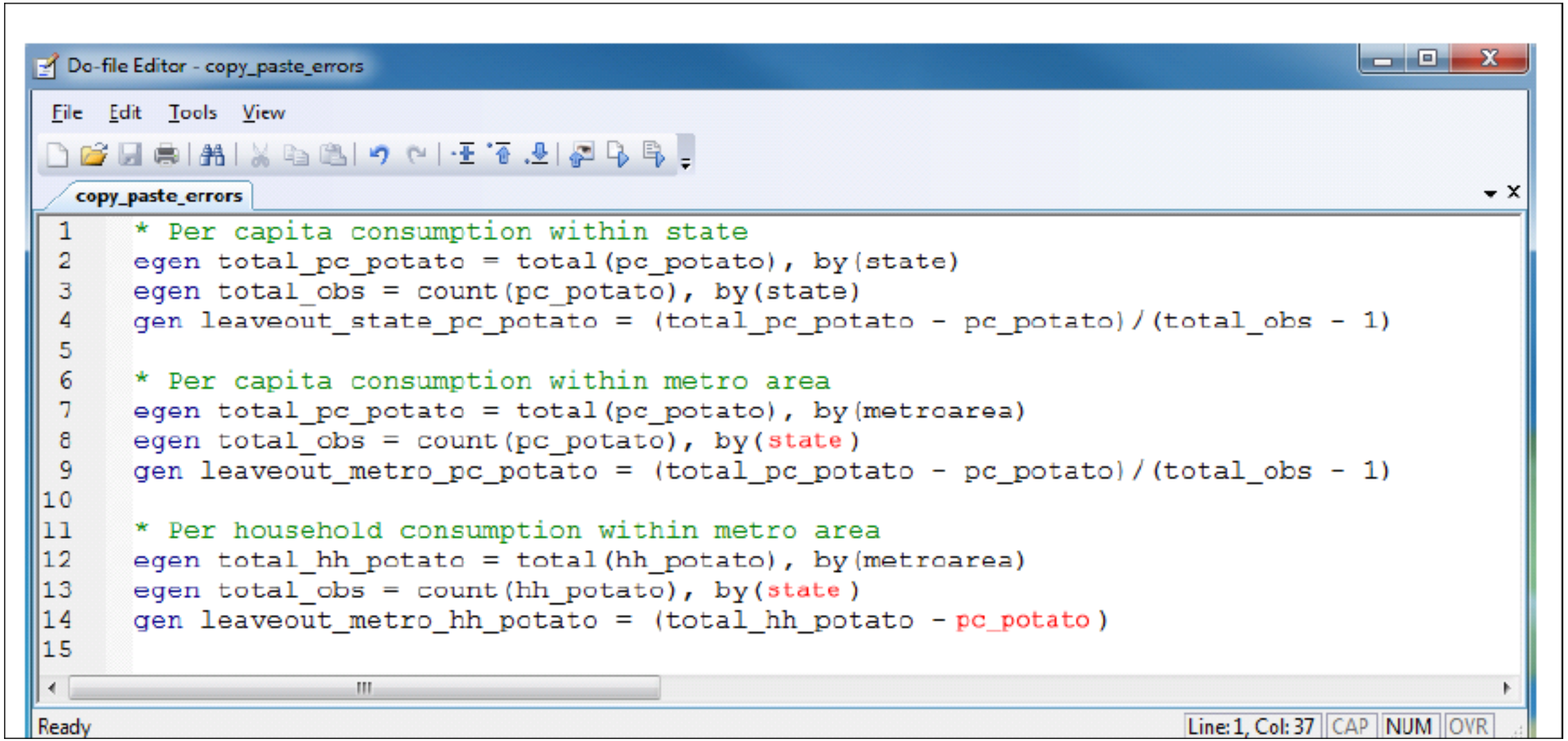


Macros reduce duplication.

```
local controls "RACE MARRIED SMSA NEWENG MIDATL ENOCENT WNOCENT SOATL ESOCENT WSOCENT MT"
local qtr "QTR120-QTR129 QTR220-QTR229 QTR320-QTR329 YR20-YR28"

use `infile'
log using `outfile', text replace

** Col 1 3 5 7 ***
reg  LWKLYWGE EDUC  YR20-YR28
reg  LWKLYWGE EDUC  YR20-YR28 AGEQ AGEQSQ
reg  LWKLYWGE EDUC  `controls' YR20-YR28
reg  LWKLYWGE EDUC  `controls' YR20-YR28 AGEQ AGEQSQ


** Col 2 4 6 8 ***
ivregress 2sls LWKLYWGE YR20-YR28 (EDUC = `qtr')
ivregress 2sls LWKLYWGE YR20-YR28 AGEQ AGEQSQ (EDUC = `qtr')
ivregress 2sls LWKLYWGE YR20-YR28 `controls'  (EDUC = `qtr')
ivregress 2sls LWKLYWGE YR20-YR28 `controls' AGEQ AGEQSQ (EDUC = `qtr')

log close
```

```
use `infile'
log using `outfile', text replace

** Col 1 3 5 7 ***
reg  LWKLYWGE EDUC  YR20-YR28
reg  LWKLYWGE EDUC  YR20-YR28 AGEQ AGEQSQ
reg  LWKLYWGE EDUC  RACE MARRIED SMSA NEWENG MIDATL ENOCENT WNOCENT SOATL ESOCENT WSOCENT MT YR20-YR28
reg  LWKLYWGE EDUC  RACE MARRIED SMSA NEWENG MIDATL ENOCENT WNOCENT SOATL ESOCENT WSOCENT MT YR20-YR28 AGEQ AGEQSQ


** Col 2 4 6 8 ***
ivregress 2sls LWKLYWGE YR20-YR28 (EDUC = QTR120-QTR129 QTR220-QTR229 QTR320-QTR329 YR20-YR28)
ivregress 2sls LWKLYWGE YR20-YR28 AGEQ AGEQSQ (EDUC = QTR120-QTR129 QTR220-QTR229 QTR320-QTR329 YR20-YR28)
ivregress 2sls LWKLYWGE YR20-YR28 RACE MARRIED SMSA NEWENG MIDATL ENOCENT WNOCENT SOATL ESOCENT WSOCENT MT  (EDUC = QTR120-QTR129 QTR220-QTR229
QTR320-QTR329 YR20-YR28)
ivregress 2sls LWKLYWGE YR20-YR28 RACE MARRIED SMSA NEWENG MIDATL ENOCENT WNOCENT SOATL ESOCENT WSOCENT MT AGEQ AGEQSQ (EDUC = QTR120-QTR129
QTR220-QTR229 QTR320-QTR329 YR20-YR28)

log close
```

Easy to make and hard to spot copy-paste errors in duplicated code



Programs remove duplication

```
program leaveout_mean
    syntax, invar(varname) outvar(name) byvar(varname)
    tempvar tot_invar count_invar
    egen `tot_invar'= total(`invar'), by(`byvar')
    egen `count_invar'= count(`invar'), by(`byvar')
    gen `outvar' = (`tot_invar' - `invar') / (`count_invar' - 1)
end

leaveout_mean, invar(pc_potato) outvar(leaveout_state_pc_potato) byvar(state)
leaveout_mean, invar(pc_potato) outvar(leaveout_metro_pc_potato) byvar(metro)
leaveout_mean, invar(hh_potato) outvar(leaveout_metro_hh_potato) byvar(metro)
```

Modularize

- Decompose the project into programs (do-files) with one function. Avoid writing programs that include more than one of these tasks:
  - modify the data,
  - create variables,
  - analyze and report results.
- Decompose programs into functions
  - A function is a reusable section of software that can be treated as a black box by the rest of the program.
  - Functions should not
    - be more than 1 page (about 60 lines) long,
    - use more than 5 or 6 input parameters,
    - reference outside information.

Programming is communication

- Document/comment – more about this below.
- Variable and function names.
- Make dependencies and requirements explicit.
- Be consistent:
  - Use a template with standard format for your dofiles.

Names

- Names should make code self-documenting.
  - bad names require comments or outside information to make sense of code

```
* sum total per capita potato consumption (tpc) by state and put in tpc_s
egen tpc_s = total(tpc), by(s)
```

  - meaningful names reveal the what the code does without comment

```
egen total_pc_potato_state = total(pc_potato), by(state)
```

- Programs and functions should have verb or verb phrase names.

Names

- Avoid one-letter names (other than counter in simple loops).

```
*Program to compute leave-out mean
program leaveout_mean
    egen total_potato= total(`1'), by(`2')
    egen total_obs= count(`1'), by(`2')
    gen `3' = (total_potato - `1') / (total_obs - 1)
    drop tot_invar count_invar
end
```

```
*Program to compute leave-out mean
program leaveout_mean
    syntax, invar(varname) outvar(name) byvar(varname)
    tempvar tot_invar count_invar
    egen `tot_invar'= total(`invar'), by(`byvar')
    egen `count_invar'= count(`invar'), by(`byvar')
    gen `outvar' = (`tot_invar' - `invar') / (`count_invar' - 1)
end
```

Be consistent: use a template

- Header.
- Set parameters and locals.
- Define programs.
- Main.
- …

State dependencies and requirements at the top of the program.

- Any external input
  that the program needs to run.
  - Data
  - ado-files
  - etc
- You can do this explicitly by commenting.
- Or by making the code self-documenting.

```
*******************************************
**** QOB Table IV
****
**** Yuqiao Huang
**** Date: May 5th 2008
**** Input: Analysis/Input/TableIV_data
**** Output: Analysis/Input/TableIV_data
*******************************************
```

```
*******************************************
**** QOB Table IV
****
**** Yuqiao Huang
**** Date: May 5th 2008
*******************************************

clear

cd $rootdir
local infile  "Analysis/Input/TableIV_data"
local outfile "Analysis/Output/TableIV.log"

use `infile'
log using `outfile', text replace
```

Programming in Stata

- Replicability
  - state Stata version at beginning of do-file
  - always set seed before random draws
- Use relative paths
  - Analysis/Code/TableIV.do
- Comments

  * Comment

  // Comment

  /* Comment */

```
*************************************
***** QOB Table IV
*****
***** Yuqiao Huang
***** Date: May 5th 2008
*************************************
version 14.1

clear
```

```
set seed 10101
gen x=runiform()
```

Documentation

- Why?
  - Replication.
  - Avoiding mistakes.
  - Efficiency: keep you on track and help with interuptions.
- Universal truths
  - It is faster to document today than tomorrow.
  - Nobody likes to write documentation.
  - Nobody regrets having written documentation.

Document everything needed for replication.

- Data sources
  - Exact data releases.
- Data management
  - How and why were variables created and cases selected, why did you dichotomize at 2 and not at 3?
- Analysis
  - What steps were taken in the data analysis, in what order, and what guided those analyses?
  - If you explored an approach and did not use it, keep a record of that as well.
- Software
  - Exact software releases used.
- Ideas and plans
  - Ideas for future research and tasks to be completed should be documented.
- …

Where?

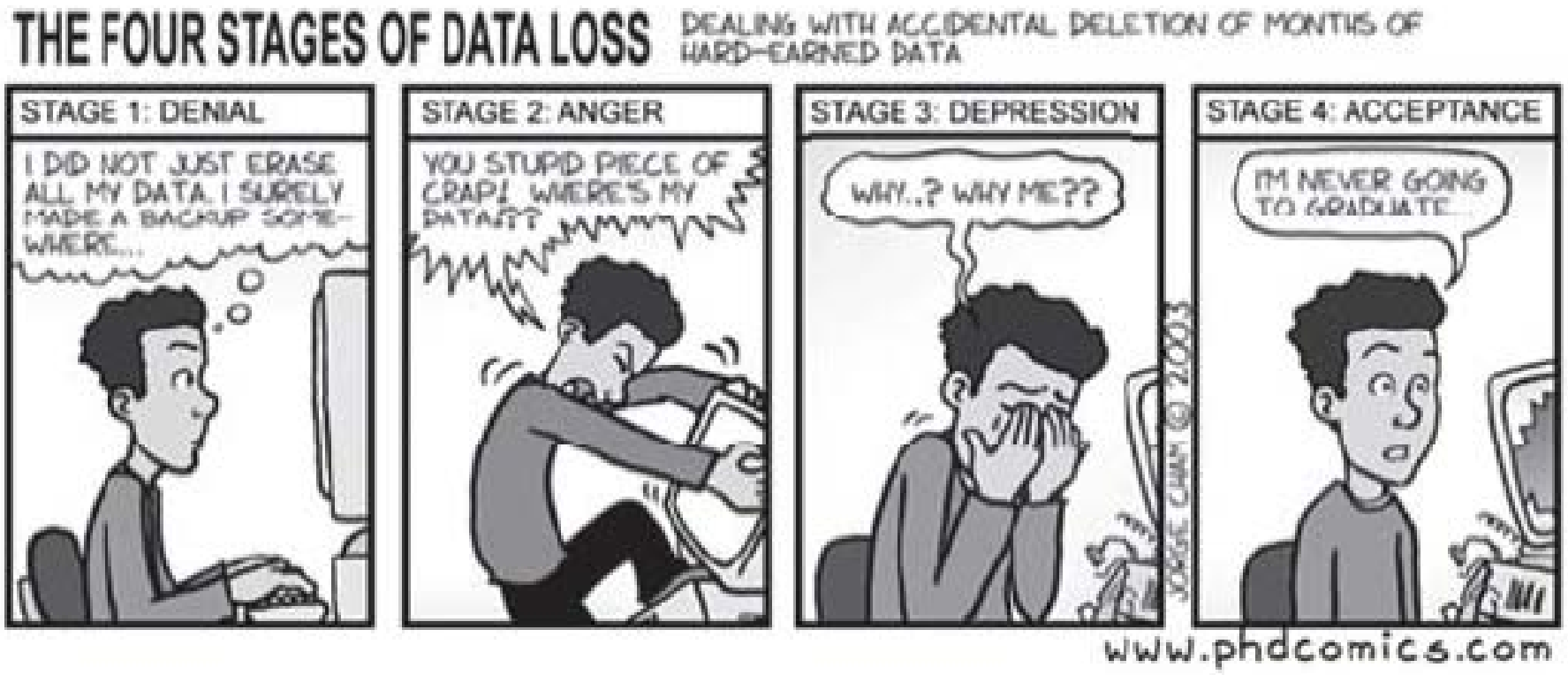
- Project diary / resarch log / lab book.
  - A chronicle of what was done, why and what was learned.
    - Log files / output of programs run with comments.
  - What was not done and why.
  - Comments from collaborators and others.
  - Plans for what to do next.
- Codebooks of the input data.
  - Description of the variables the project started with.
  - Metadata about the datasets and variables.
- Project map and **dofiles/programs**
  - Programs should be commented and self-documenting.

Resarch log: example

- Observability_spurious_spread.rtf in CA\Logs\Documents\

Backup and Version Control

Have a backup plan



Accidental deletion, hardware failure, viruses, theft, power surges, etc.

Have a backup plan

- What to back up.
- Where.
- How frequently: instantaneously, when saving, daily, long-term.
- Keep track of what is backed up.
- What tools to use.

What to back up

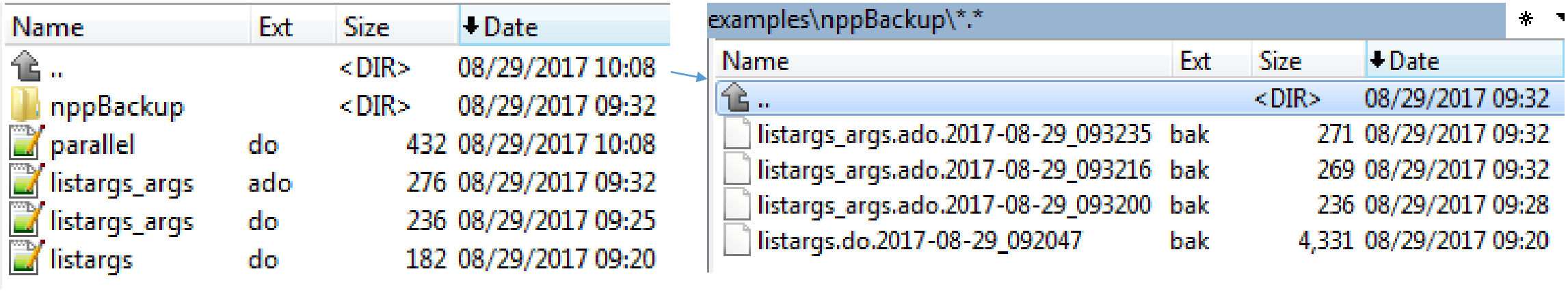
- Long-time storage
  - How
    - Archives: check with your university for storage solutions.
  - What
    - Raw data.
      - Ensure to have raw data backed up in more than one location.
    - Final project
      - For replication and documentation.

Short-term storage

- Instantly
  - mirror
  - protects against disk crash.
- Every time you save a file.
  - Back up (almost) everything created by a human being as soon as it is created.
    - verbose backup
  - protects against short term errors, helps reverting.
- Daily
  - protects against disk crashes and helps reverting.
- Version control
  - Save whole project intermittently.
  - Preserve permanently

Verbose backup using Notepad++

- Everytime I save a file, Notepad++ saves a timestamped copy of the previously saved file in a subdirectory called nppBackup
  - I have copies of all programs edited by Notepad++
    - Stata-dofiles, Python, R, Perl etc.,



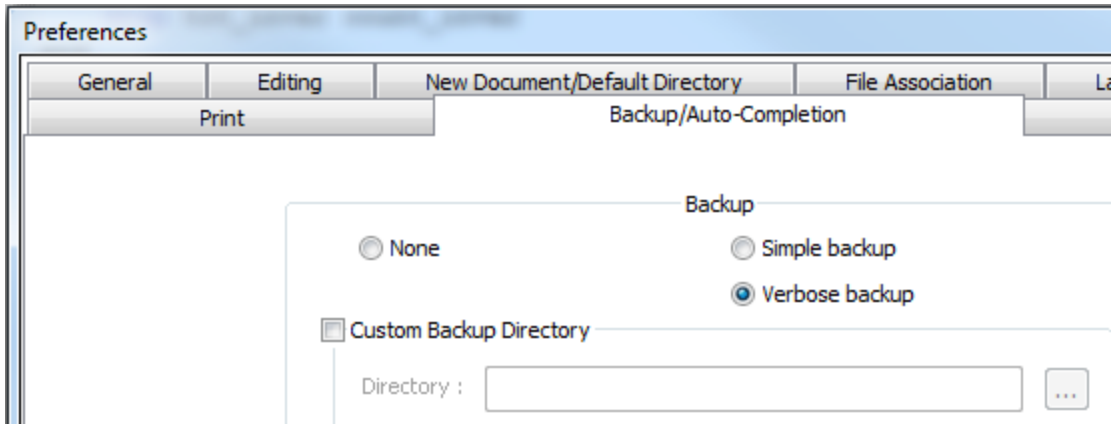
Name	Ext	Size	Date
..		<DIR>	08/29/2017 10:08
nppBackup		<DIR>	08/29/2017 09:32
parallel	do	432	08/29/2017 10:08
listargs_args	ado	276	08/29/2017 09:32
listargs_args	do	236	08/29/2017 09:25
listargs	do	182	08/29/2017 09:20

examples\nppBackup*.*

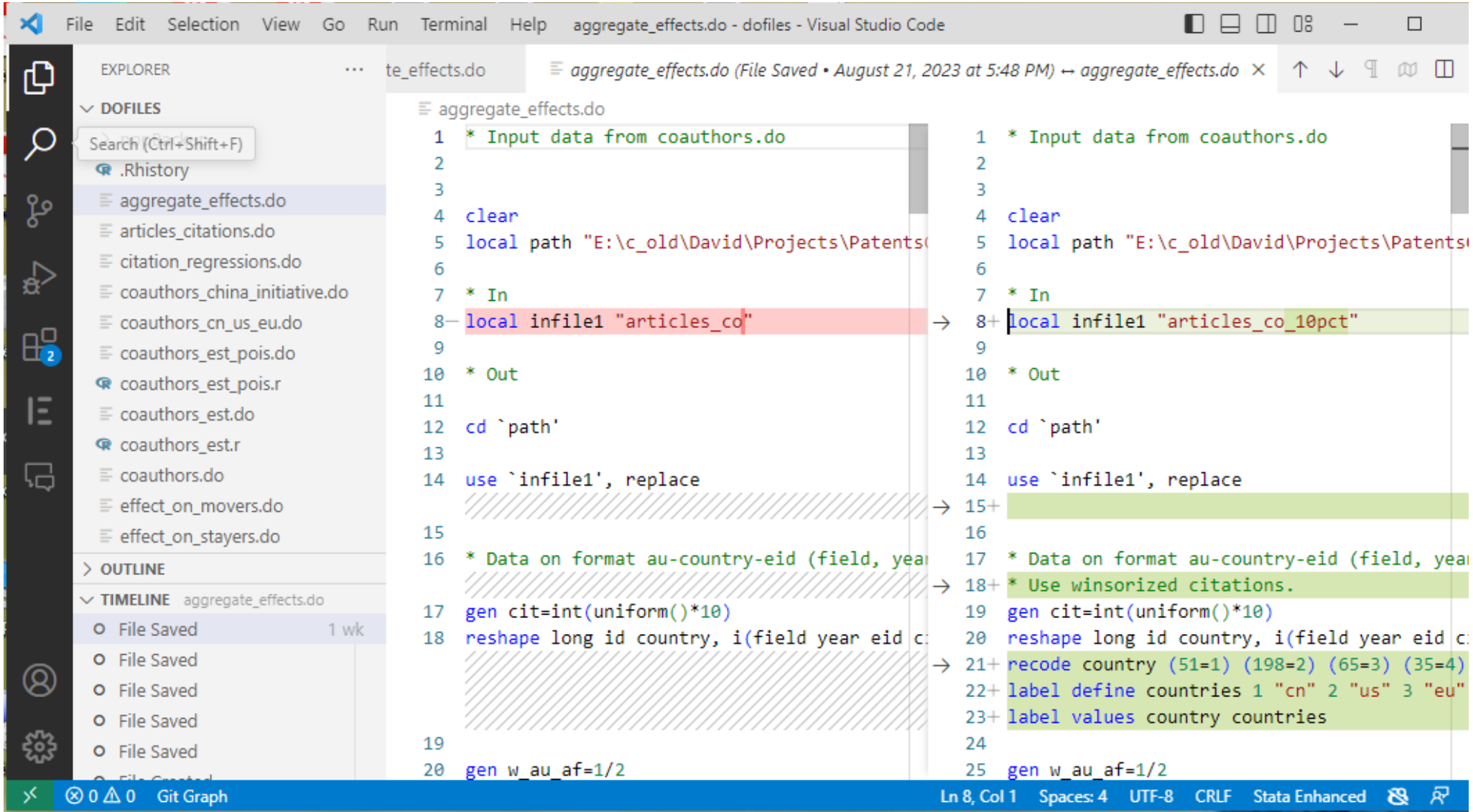
Name	Ext	Size	Date
..		<DIR>	08/29/2017 09:32
listargs_args.ado.2017-08-29_093235	bak	271	08/29/2017 09:32
listargs_args.ado.2017-08-29_093216	bak	269	08/29/2017 09:32
listargs_args.ado.2017-08-29_093200	bak	236	08/29/2017 09:28
listargs.do.2017-08-29_092047	bak	4,331	08/29/2017 09:20

Verbose backup using Notepad++

- Turn on verbose backup in Settings/Preferences/
- If you prefer, you can use a custom backup directory (rather than nppBackup)



Visual Studio Code Timeline



Daily backup

- Back up entire working hard drive daily, to two alternating locations.
  - If something fails during backkup, there is still a functioning copy.
  - You can always revert to a copy saved at least one day ago.

A simple backup system

1. Robocopy – Windows command
   - robocopy <Source> <Destination> [<File>[...]] [<Options>]
   - Options
     - /e Copies subdirectories. Note that this option includes empty directories.
     - /mir Copies subdirectories. Note that this option includes empty directories. Deletes destination files and directories that no longer exist in the source.
     - /np Specifies that the progress of the copying operation (the number of files or directories copied so far) will not be displayed.
     - /log:<LogFile> Writes the status output to the log file (overwrites the existing log file).

backup.bat | AK91_log.txt

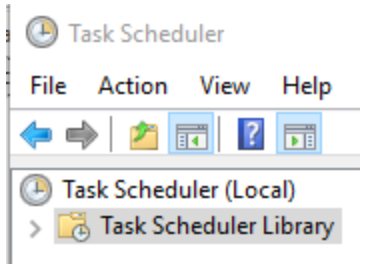
```
1  robocopy e:\AK91 g:\AK91 /mir /np /log:D:\backup\AK91_log.txt
2  pause
```

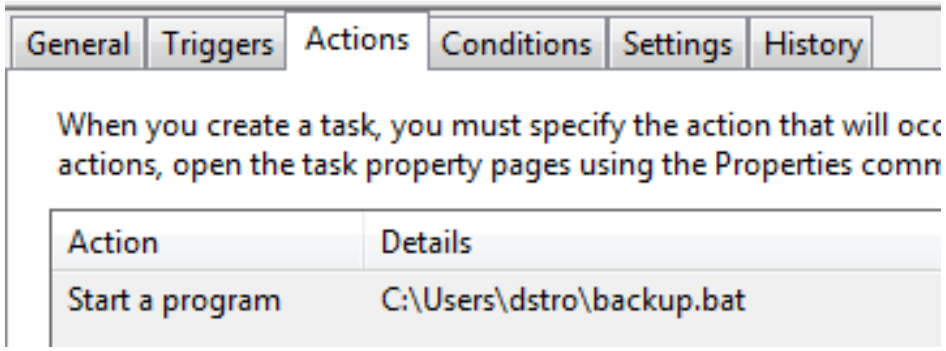
AK91_to_G.bat | AK91_log.txt

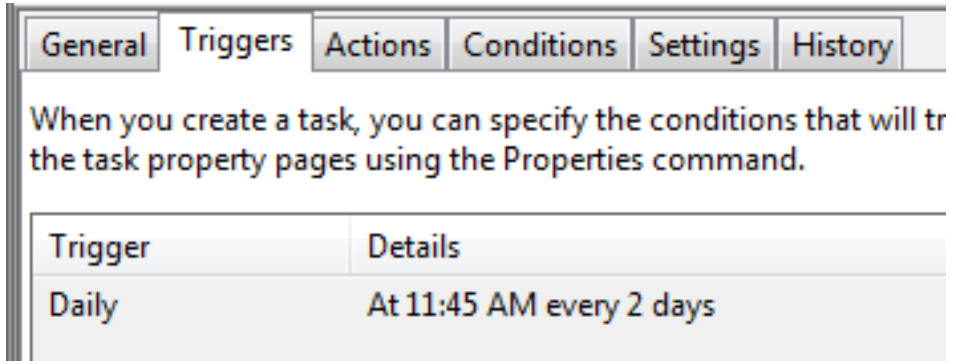
```
1
2  ------------------------------------------------------------------------------
3    ROBOCOPY     ::     Robust File Copy for Windows
4  ------------------------------------------------------------------------------
5
6    Started : Wednesday, 29 August 2018 14:04:51
7     Source = e:\AK91\
8       Dest : g:\AK91\
9
10     Files : *.*
11
12   Options : *.* /S /E /DCOPY:DA /COPY:DAT /PURGE /MIR /NP /R:1000000 /W:30
13
14 ------------------------------------------------------------------------------
15
16       New Dir          3    e:\AK91\
17         New File              15    .gitignore
```

A simple backup system

2. Use Task Scheduler to execute the .bat file every two days.







Task 2a: Clean Data Procedure

Angrist and Krueger (1991) and Angrist and Lavy (1999)

Raw data to input data. First set up folder structure. Then:

1. Import the raw data into a raw data folder.
   Deny writing to this folder.
2. Normalize the data set AL99
   rename variables and save in Build/Input folder.
3. Write a program that does a values review of the AK91 and AL99 data.
   - Loop over datasets
     - Loop over variable types (string, integer, float)
       - Do a values review for each type and print to an output file

Task 2b: Clean AK91 code.

- Clean the code of AK91 that you created in Task 1.
  - Review names.
  - Remove duplication of code and data.
    - First use macros and loops.
    - Then use the xi command instead of some loops.
  - Print output regression tables.
  - Write a program that runs regressions and prints output tables of a subset of the x-variables.
    - Call this program to create columns 1 3 5 7 in the tables.