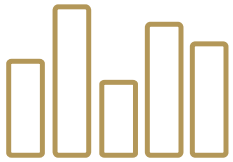


STRATEGIC DATA PROJECT

SDP STATA GLOSSARY

SDP TOOLKIT

FOR EFFECTIVE DATA USE IN EDUCATION AGENCIES

www.gse.harvard.edu/sdp/toolkit

Toolkit Documents

An Introduction to the SDP Toolkit for Effective Data Use

Identify: Data Specification Guide

Clean: Data Building Guide for College-Going
Clean: Data Building Guide for Human Capital BETA

Connect: Data Linking Guide for College-Going
Connect: Data Linking Guide for Human Capital BETA

Analyze: College-Going Success Analysis Guide
Analyze: Human Capital Analysis Guide BETA

Adopt: Coding Style Guide

SDP Stata Glossary

VERSION: 1.2
Last Modified: September 2, 2013

| Authored by Todd Kawakita and Eniko Nagy

STATA GLOSSARY

Brush up on Stata using this glossary of commonly used commands within the toolkit. The glossary outlines many useful commands and functions relevant to data cleaning and exploration. The glossary is provided in either alphabetical order or by topic. If you click on a command/function in the alphabetical glossary, it will take you to the topical glossary which includes examples.

Remember that you can type commands directly into the Stata window; however, in most cases, you will find it useful to collect a set of commands into a do file. A do file is a document that allows us to run the code multiple times during development, and allows us to save our work for future use.

Alphabetical Stata Glossary

append - Appends stored datasets to the end of the dataset in memory.

assert - Allows us to test if something is true about the data.

bysort - Organizes the data by certain variables and executes a command.

browse - Browse using the data editor.

capture - Executes command, suppressing all its output (including error messages, if any) and issues a return code of zero.

cd - Change directory.

center - Standardizes a variable.

clear - Clears the current dataset from memory.

codebook - Describes our data content for variables in the dataset.

collapse - Converts our dataset into a dataset of means, sums, medians, etc.

count - Counts the observations that fulfill a condition that we specify.

date() - Returns the number of days since 01jan1960 based on a string description of a date.

describe - Describes the current memory.

destring - Changes the data type from "string" or letter format to a numeric format.

display - Displays a message in the Stata window.

drop - Drops all variables or observations listed and keeps all others. You can combine the drop command with if, to specify criteria for dropping certain observations.

duplicates - Tags, reports or drops duplicates observations.

egen - Generates variables which are functions of other variables.

foreach - Sets a local macro to each element in a list and executes commands enclosed in braces. The loop is executed the number of times the code specifies.

forvalues - Sets a macro to each element of a numeric list and executes commands enclosed in braces.

format - Sets the output format for the variable.

generate - Generates new variables and populates with data.

global - Assigns strings to a global macro name.

gsort - Allows you to sort data in both ascending and descending order.

help - Displays online help for a Stata command or function.

inlist() - returns 1 if z is a member of the remaining arguments; otherwise, it returns 0.

insheet - Reads into memory text data created by a spreadsheet or database program other than Stata.

isid - Checks if data is unique by a listed variable().

keep - Keeps only variables or observations listed and drops all others. One can combine the **keep** command with **if**, to specify criteria to keep certain observations.

list - Lists values of selected variables.

label - Labels variables or values of variables.

levelsof - displays a sorted list of the distinct values of a variable.

local- Assigns strings to local macro names.

lower() - Returns a lower case version of a string.

max() - Returns the maximum value of a specified variable or number.

merge - Combines datasets. The datasets must share common variables that are unique in at least one dataset.

mdy() - Returns the days since 01jan1960 after inputting (M,D,Y) where M is (1 -12) D is (1 - 31) and Y is the year.

min() - Returns the minimum value of a specified variable or number.

missing - Returns "1" if an argument is missing, returns a 0 if it is not missing.

_n - Contains the number of current observation.

_N - Contains the total number of observations in the database.

nvals() - Returns the number of distinct values of a variable.

odbc - Load, write, or view data from ODBC sources.

order - Reorders selected variables in a dataset.

preserve - **Preserve** saves the data in its current state to allow data manipulation until **restore** is used to restore the data to its former state when the **preserve** command was used. When running a .do file, it is important that one does not stop the file in the middle of a preserve – restore sequence, or else the data will revert back to its state when the preserve was entered.

proper() - Returns a string with the first letter of every word capitalized.

regexm() - Performs a match of a regular expression and evaluates to 1 if regular expression re is satisfied by the string s, otherwise returns 0.

regexr() - Replaces the first substring within s1 that matches re with s2 and returns the resulting string.

replace - Replaces values of a variable.

rename - Changes the name of an existing variable.

renvars - Renames multiple variables.

reshape - Restructures the dataset "wide" or "long".

restore - preserve saves the data in its current state to allow data manipulation until **restore** is used to restore the data to its former state when the **preserve** command was used.

round() - Returns values rounded to the nearest integer (unless specified otherwise).

save - Saves a Stata dataset.

sort - Sorts data in ascending order under variables listed.

summarize - Describes our data content for variables in the dataset.

tabulate - Examines the distribution of values for a variable. This command is essential for checking data.

- A tab with one variable shows frequencies for each value of the variable.
- A tab with two variables (often called a cross-tab) produces a matrix of frequencies for the values of one variable against the values of another variable.

tempfile - Creates a convenient temporary file that can be referenced at different points in the program.

tostring - Changes a variable from numeric to string format.

trim() - Removes trailing spaces from a string.

quietly - Suppresses output for the duration of the command.

xtile - Creates 'quantile' categories for a variable.

upper() - Returns an upper case version of a string.

use - Loads a Stata dataset.

!= or ~= - Indicates a variable is not equal to something else.

>, <, >=, <=, == - Operators that describe a variable as greater than, less than, greater than or equal to , less than or equal to, or equal to.

&, | - Logical operators describing "and" and "or", respectively.

GETTING HELP

help - Displays online help for a Stata command or function.

To get help with any Stata command or function, type "help" followed by the command/function name in the main Stata window's Command section.

Example:

```
help summarize
```

ABBREVIATIONS

Stata recognizes abbreviated commands, so for frequently used commands you do not have to always type out the full word of the command. Some of the most common abbreviations include:

```
generate  gen
missing   m
tabulate  tab
display   di
```

INSERTING COMMENTS IN YOUR DO FILE

When writing a do file, it is useful to insert frequent comments that describe what your code is doing. Please refer to **Adopt**: Coding Style for the recommended practices for inserting comments.

Comment lines start with `*` or `//`; this tells Stata to not execute the line.

`//` can also follow at the end of the line after a Stata command.

Block comments that extend over multiple lines can be enclosed in `/* . . . */`

Example:

```
* This is a comment line

// this is also a comment line
```

```
gen district = "MyDistrict"   // this is a comment.

/* This is a comment line
that extends on multiple lines */
```

BREAKING UP LONG LINES OF CODE

If a code is long, it can be practical to break it into multiple lines. Line breaks are indicated by /// at the end of the line; this tells Stata that the code continues on the next line.

Another way to break long lines is to enclose the code between #delimit ; and #delimit cr . This is useful for graph commands, or for pulling in datasets through ODBC connections; both of these tend to be very long command lines.

Example:

```
replace math_flag = 1 if regexm(course_code_desc, "GEOM") | ///
      regexm(course_code_desc, "MATH") | ///
      regexm(course_code_desc, "ALGEBRA")

#delimit ;
replace math_flag = 1 if regexm(course_code_desc, "GEOM") |
      regexm(course_code_desc, "MATH")  |
      regexm(course_code_desc, "ALGEBRA");
#delimit cr
```

QUOTES

Stata uses three sets of quotes:

Single quotes: used mainly to refer to local macros (read more about macros in Section 4). It is important to note that the left single quote is the character typically located at the top left of your keyboard, near the number 1. The right single quote is the standard single quote, or apostrophe. (`mylocal')

Double quotes: used to enclose strings. ("yes")

Compound double quotes: the compound double quote uses the same left and right single quote as described above, and a pair of double quotes (`"example"'). They operate the same way as double quotes, but in case the string itself contains double quotes, a compound double quote is needed to indicate that the string contains quotation marks.

Example:

```
local gend = 1
replace building = 1 if gender == `gend'
// This is a single quote, referring to a macro

local race = "asian"
// This is  a double quote enclosing a string
generate group = 3 if race_ethnicity == "`race'"
// First, the local has to be enclosed in single quotation makrs to indicate that it is a macro.
Then, because the local refers to a string, it has to be enclosed in double quotation marks.

generate myvar = `" "line 1" "line 2" "'
// The string variable created will have the value "line 1" "line 2". This can be helpful when
creating variables that you want to display in graphs; the quotes allow the text to be broken into
separate lines.
```

1. DATA MANAGEMENT

1.1 Handling Datasets

`cd ["]drive:directory_name["]` – Change directory.

Allows you to specify the working directory. This is useful as a first command in a do file to ensure that you are working in the directory that your data is located.

If your directory_name contains embedded spaces, remember to enclose it in double quotes.

Example:

```
cd "C:\My Stata Folder"
```

`clear [all]` – Clears the current dataset from memory.

`clear`, by itself, removes data and value labels from memory. It is typical to use the form `clear all`, which removes all objects from memory.

`clear` can also be added as an option to a command that loads a dataset into Stata.

Example:

```
clear all

use Student_Attributes.dta, clear
```

`use filename [, clear]` - Loads a Stata dataset.

`use` loads into memory a Stata-format dataset previously saved by save. If filename is specified without an extension, .dta is assumed. If your filename contains embedded spaces, remember to enclose it in double quotes.

Example:

```
use Student_Attributes.dta
use "My Data File", clear
```

`insheet using filename [, options]` - Reads into memory text data created by a spreadsheet or database program other than Stata.

If the data you are trying to load is created by another database program than Stata (e.g. csv or text file), you cannot use the use command. Instead, you must use insheet. Insheet reads text (ASCII) files in which there is 1 observation per line and the values are separated by tabs or commas.

The most common options used with `insheet` are:

`comma` – specifies that your data is comma delimited
`tab` – specifies that your data is tab delimited
`clear` – replaces data in memory, as described above
`[no]names` – specifies if variable names are included on the first line of the file.

Example:

```
insheet using "My CSV file.csv", names
```

odbc - Load, write, or view data from ODBC sources.

If you have ODBC connection(s) set up to connect directly to your agency's data source, you can use the `odbc` command to load data into Stata directly from the data source. You can also use `odbc list` to list the odbc connections you have set up.

Within the `odbc` command you use standard SQL code to load data from a table.

Example:

```
odbc load, clear exec("SELECT school_year, loc_id FROM schools") dsn(your_odbc_connection)
```

merge (1:1 | m:1 | 1:m) varlist using filename [,options] - Combines datasets. The datasets must share common variables that are unique in at least one dataset.

Using `merge`, one can link datasets in a one-to-one (`1:1`), or one-to-many relationship (`1:m | m:1`). A one-to-one relationship occurs when the variables used to link datasets are unique in both datasets. A one-to-many relationship occurs when the variable is unique in only one of the datasets. By default, a _merge variable is created, showing whether the values of the linking variable exist only in the master dataset (`data_set_1` below), using dataset (`data_set_2` below), or matched (exist in both datasets). A tab of the `_merge` variable will help one grasp how well the datasets merged.

Note, that Stata can only merge datasets that are in Stata format; if the data you are trying to merge is in another format (for example, csv), first you have to load it to Stata and save it as a Stata file or as a temporary file (see below).

Example:

```
use data_set_1
merge 1:m sid using data_set_2
```

data_set_1
(unique by sid)

sid	gender
1	male
2	female
3	male
5	female

data_set_2

sid	subject	test_score
1	math	85
1	ela	75
2	math	90
2	ela	78
3	math	86
3	ela	68
4	math	58
4	ela	62

→

merged dataset

sid	subject	test_score	gender	_merge
1	math	85	male	matched (3)
1	ela	75	male	matched (3)
2	math	90	female	matched (3)
2	ela	78	female	matched (3)
3	math	86	male	matched (3)
3	ela	68	male	matched (3)
4	math	58		using only (2)
4	ela	62		using only (2)
5		.	female	master only(1)

Common options used with `merge` are:

`keepusing(varlist)` - variables to keep from using data; default is all
`generate(newvar)` - name of new variable to mark merge results; default is _merge
`nogenerate` - do not create _merge variable
`keep(1 | 2 | 3)` - specify which match results to keep. This is based on the _merge variable.

Example:

```
merge 1:m sid using dataset_2, keepusing(test_score)
// specifies that from dataset_2 only test_score will be merged to the dataset in memory, and all
other variables (in our case, subject) will be ignored

merge 1:m sid using dataset_2, keep(1 3)
// specifies that observations that appear in both datasets will be kept in the merged data
(indicated by the value 3); additionally, all observations that appear in the data in memory but not
in dataset_2 are kept (indicated by the value 1).
```

`append using filename [, options]` - Appends stored datasets to the end of the dataset in memory.

`append` appends Stata-format datasets stored on disk to the end of the dataset in memory. If any filename is specified without an extension, .dta is assumed.

Variables in the appended dataset that are named the same as variables in the dataset will be treated as the same, and populated under the same variable (column). Variables that have different names will be added as new variables.

Note, that Stata can only merge datasets that are in Stata format; if the data you are trying to merge is in another format (for example, csv), first you have to load it to Stata and save it as a Stata file or as a temporary file (see below).

Example:

```
use `data_set_1'
append using `data_set_2'
```

data_set_1

sid	subject
1	Math
2	ELA
3	ELA

data_set_2

sid	test_score	school_year
3	83	2006
4	50	2007

→

appended_dataset

sid	subject	test_score	school_year
1	Math	79	
2	ELA	81	
3	ELA	65	
3		83	2006
4		50	2007

save [filename] [, save_options] – Saves a Stata dataset.

save stores the dataset currently in memory on disk under the name filename.

The most common option is replace, which allows Stata to overwrite a file that might exist with the same file name in the current folder. This is useful if you are developing code and will run your do file multiple times.

The filename can also contain a path to a specific folder. If the folder name or file name contains embedded spaces, remember to enclose it in double quotes.

Example:

```
save School
save "folder\School.dta", replace
```

1.2 Structuring Datasets

preserve – Preserve saves the data in its current state to allow data manipulation until restore is used.

Restore restores the data to its former state when the preserve command was used. In a .do file, it is important that one does not run the file from preserve to somewhere in the middle of a preserve – restore sequence: the data will revert back to its state when the preserve was entered.

restore – Restores the data to its former state when the preserve command was used.

preserve – restore can be used when you want to manipulate your data temporarily, but want to revert back to the original data once your temporary operation is complete. For example, you might want to count the unique students in your data, but then revert to the data in memory with all variables and observations.

Example:

```
preserve
     keep sid
     duplicates drop
     count
restore
```

tempfile - Creates a convenient temporary file that can be referenced at different points in the program.

`tempfile` creates a local macro in which the dataset in memory is used. References to this macro are done using local macro conventions (read more in Section 4).

A common use of `tempfile` is to save a data set in memory that was loaded from a data file different from a Stata file.

Example:

```
insheet using "mydata.csv"
tempfile mydata_stata
save `mydata_stata'
use statafile, clear
merge 1:1 sid using `mydata_stata'
```

Another way to accomplish the example listed under `preserve - restore` is as follows:

```
tempfile mydata
save `mydata'
keep sid
duplicates drop
count
use `mydata', clear
```

collapse [(stat)] [target_var =] varlist [, options] - Converts our dataset into a dataset of means, sums, medians, etc.

Using collapse, you can create a dataset of summary statistics by certain characteristics of the data.

`stat` refers to the statistical function you want to apply to the collapse. If none is specified, `mean` is assumed. The most common statistical functions used are `mean, median, count, sum, min, max`.

By specifying a `target_var`, you can rename your variable in in the resulting dataset.

The most typical option that we use is `by`, which describes the characteristic you want to collapse your data by. This can include one or more variables.

Note that variables not included in collapse will be dropped from your data.

Example:

```
// Assume you want to get a count of students in each high school, and their average SAT score.
collapse (count) N=sid (mean) SAT_score, by(school_name)
```

initial dataset

sid	gender	SAT_score	school_name
1	male	1000	School A
2	male	1100	School A
3	female	1050	School A
4	male	1100	School B
5	female	1200	School B

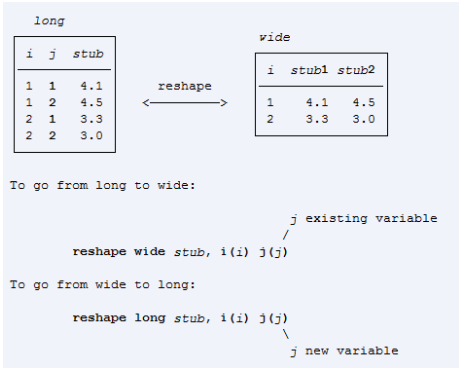
→

collapsed dataset

N	SAT_score	school_name
3	1050	School A
2	1150	School B

`reshape wide | long stub, i(i), j(j)` - Restructures the dataset "wide" or "long".

A wide dataset is typically unique by an id, such as sid, with data for each year varying across the same row. A long dataset would have an id listed multiple times with each year listed on a different row. To the right is the proper syntax for the reshape command, supplied from the help menu in Stata.



Example:

```
reshape wide test_score, i(sid) j(school_year)
// dataset will now be unique by sid
isid sid
```

data in long format

sid	school_year	test_score
1	2007	85
1	2008	75
2	2007	90
2	2008	78

i j stub

→

data in wide format

sid	test_score2007	test_score2008
1	85	75
2	90	78

i stubj stubj

```
reshape long test_score, i(sid) j(school_year)
```

data in wide format

sid	test_score2007	test_score2008
1	85	75
2	90	78

i stubj stubj

→

data in long format

sid	school_year	test_score
1	2007	85
1	2008	75
2	2007	90
2	2008	78

i j stub

2 VARIABLE MANAGEMENT

2.1 Handling Variables

drop {varlist | drop if expression} - Drops all variables or observations listed and keeps all others. You can combine the drop command with if, to specify criteria for dropping certain observations.

Example:

```
drop gender race_ethnicity
// drops the variables gender and race_ethinicity from the dataset

drop if gender == "male"
// drops all males from the dataset
```

keep {varlist | keep if expression} - Keeps only variables or observations listed and drops all others. One can combine the keep command with if, to specify criteria to keep certain observations.

Example:

```
keep sid school_year school_name
// keeps only sid, school_year and school_name. All other variables are dropped
keep if gender == "female" | mi( gender)
// keeps only females, or observations where gender is missing.
```

generate [type] newvar =exp [if] - Generates new variables and populates with data.

Using `type`, you can specify the type of the variable created. Type can be `byte|int|long|float|double|str|str1|str2|...|str244`. If `type` is not specified, Stata automatically determines the type based on the expression specified. Generally you do not need to specify the type of your variables.

If you specify a condition using `if`, observations that do not meet the criteria will be populated by a missing value.

Example:

```
generate district = "MyDistrict"
generate building = 1 if gender == "male"
```

original dataset

sid	gender
1	male
2	female
3	male

→ resulting dataset

sid	gender	district	building
1	male	MyDistrict	1
2	female	MyDistrict	.
3	male	MyDistrict	1

egen newvar = fcn(arguments) [, by()] - Generates variables which are functions of other variables.

`egen` creates equal to `fcn(arguments)`. Here `fcn()` is a function specifically written for `egen`, as documented below or as written by users.

Using `generate` (discussed above) you can only specify criteria that refers to the observation in question. In contract, `egen` can scan multiple observations and populate a new variable based on values of multiple observations.

The most common functions used are `count`, `min`, `max`, `mean`, `median`, `mode`, `rowmax`, `rowmean`, `rowmiss`, `rownonmiss` and `concat`.

Example:

```
egen max_SAT_score = max(SAT_score), by(sid)
// takes the maximum SAT score obtained by a student and populates this value across all
observations

gen space = ", "
egen fullname = concat(lastname space firstname)
drop space
```

original dataset

sid	school_year	last_name	first_name	SAT_score
1	2005	Smith	Maria	1050
1	2006	Smith	Maria	1100
1	2007	Smith	Maria	1080

→ resulting dataset

sid	school_year	last_name	first_name	SAT_score	max_SAT_score	fullname
1	2005	Smith	Maria	1050	1100	Smith, Maria
1	2006	Smith	Maria	1100	1100	Smith, Maria
1	2007	Smith	Maria	1080	1100	Smith, Maria

duplicates - Tags, reports or drops duplicates observations.

duplicates report [varlist] - reports duplicates in terms of all variables or the variables specified
duplicates tag [varlist], generate(newvar) - creates a new variable and populates it with the number of surplus observations in terms of all variables or variables specified
duplicates drop varlist [,force] - drops duplicates in terms of all variables or the variables specified. The force option is required when such a varlist is given as a reminder that information may be lost by dropping observations, given that those observations may differ on any variable not included in varlist.

Example:

sid	school_year
1	2005
1	2005
1	2006

duplicates report

Duplicates in terms of all variables

copies	observations	surplus
1	1	0
2	2	1

duplicates tag sid school_year, gen(dup)

sid	school_year	dup
1	2005	1
1	2005	1
1	2006	0

duplicates drop

sid	school_year	dup
1	2005	1
1	2006	0

replace oldvar =exp [if] - Replaces values of a variable.

Example:

```
replace building = 2 if gender == "female"
```

original dataset

sid	gender	district	building
1	male	MyDistrict	1
2	female	MyDistrict	.
3	male	MyDistrict	1

resulting dataset

sid	gender	district	building
1	male	MyDistrict	1
2	female	MyDistrict	2
3	male	MyDistrict	1

destring [varlist] , {generate(newvarlist)|replace} [force] [ignore()] - Changes the data type from "string" or letter format to a numeric format.

Variables in varlist that are already numeric will not be changed. `destring` treats both empty strings "" and "." as missing values. `destring` ignores trailing spaces.

If the string contains non-numeric characters and `force` is not specified, Stata returns an error. If `force` is specified, Stata populates the new variable with missing values where a non-numeric character is found.

`ignore` tells Stata to ignore the string specified and convert the value as if the specified string did not exist in it.

Example:

```
destring SAT_score, gen(SAT_score_numeric), force ignore("pt")
```

original dataset

sid	SAT_score
1	"1050"
2	"1100 "
3	""
4	"1080!"
5	"pt1200"

resulting dataset

sid	SAT_score	SAT_score_numeric
1	"1050"	1050
2	"1100 "	1100
3	""	.
4	"1080!"	.
5	"pt1200"	1200

tostring varlist , {generate(newvarlist)|replace} - Changes a variable from numeric to string format.

When converting to string, the most compact string format possible is used.

Example:
tostring SAT_score, replace

original dataset

sid	SAT_score
1	1050
2	1100
3	.

→

resulting dataset

sid	SAT_score
1	"1050"
2	"1100"
3	""

2.2 Summarizing Variables

summarize [varlist] [, options] - Describes our data content for variables in the dataset.

Example:

```
summarize male
```

```
    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
        male |     21803    .5013072    .5000098          0          1
```

codebook [varlist] - Describes our data content for variables in the dataset.

If no variables are specified, codebook returns a description of all variables in the dataset.

Example:

```
use School.dta
codebook school_year
```

```
-------------------------------------------------------------------------------
school_year
(unlabeled)
-------------------------------------------------------------------------------

                        type:         numeric (double)

                       range:     [2004,2009]                         units:  1
                unique values:     6                        missing .:  0/191

                   tabulation:    Freq.      Value
                                  35         2004
                                  38         2005
                                  38         2006
                                  38         2007
                                  31         2008
                                  11         2009
```

describe [varlist][, fullnames] – Describes the current memory.

Provides a description of the dataset. If `varlist` is specified, only the variables specified will be included in the output. `fullnames` tells Stata to not abbreviate the variable names. This is useful if you would like to distinguish between long variable names that are similar in the data set.

Example:

```
use School.dta
describe

Contains data from  School.dta
  obs:           191
 vars:             7                          9 Jun 2012 03:02
 size:        13,943
-------------------------------------------------------------------------------
                    storage   display     value
variable name       type           format      label        variable label
-------------------------------------------------------------------------------
```

```
school_year         double          %10.0g
school_code         double          %10.0g                  school_code
school_name         str40           %40s                    school_name
grade_span          str5            %9s
middle              float           %9.0g
high                float           %9.0g
elementary          float           %9.0g
-------------------------------------------------------------------------------
Sorted by:  school_year  school_code
```

browse [varlist] [if] [in] [, nolabel] - Browse using the data editor.

Example:

```
browse male
```

tabulate - Examines the distribution of values for a variable. This command is essential for checking data.

tabulate varname [if] [,missing] - A tab with one variable shows frequencies for each value of the variable.

tab1 varlist [if] [,missing] – Displays one way tables for each of the variables included in varlist

tabulate varname1 varname2 [if] [,missing] [row] [col] - A tab with two variables (often called a cross-tab) produces a matrix of frequencies for the values of one variable against the values of another variable. Specifying row and/or col adds percentages by row and/or column.

If the `,mi` option is added, the tabulation will include missing values in the tabulation.

Example: examining the gender distribution within the dataset.

```
. tab male, mi

       male |      Freq.     Percent        Cum.
------------+-----------------------------------
          . |          1       20.00       20.00
          0 |          2       40.00       60.00
          1 |          2       40.00      100.00
------------+-----------------------------------
      Total |          5      100.00
```

Example: examining which students qualify for free lunch.

```
 . tab sid frpl

           |         frpl
       sid |         0          1 |     Total
-----------+----------------------+----------
         1 |         0          1 |         1
         2 |         0          1 |         1
         3 |         0          1 |         1
         4 |         0          1 |         1
         5 |         1          0 |         1
-----------+----------------------+----------
     Total |         1          4 |         5
```

count [if] - Counts the observations that fulfill a condition that we specify.

Example:

```
count
//counts all observations in the dataset

sort sid
count if sid != sid[_n-1]
// counts unique student ids
```

nvals() - Returns the number of distinct values of a variable.

Note: egenmore is a free package you must install for Stata. The package can be installed using the following command: ssc install egenmore and adds on the nvals egen function.

nvals is used together with egen to count the number of unique observations with a certain criteria.

Example:

```
bys sid: egen nvals_test_score = nvals(test_score)
```

original dataset

sid	SAT_score
1	1050
1	1100
2	1010

resulting dataset

sid	SAT_score	nvals_SAT_score
1	1050	2
1	1100	2
2	1010	1

2.3 Organizing Variables

sort varlist - Sorts data in ascending order under variables listed.

sort arranges the observations of the current data into ascending order based on the values of the variables in varlist. There is no limit to the number of variables in the varlist. Missing numeric values (see missing) are interpreted as being larger than any other number, so they are placed last.

Example:

```
sort sid school_year school_name
// sorts the data first by sid, then school_year then school_name
```

gsort [+|-] varname - Allows you to sort data in both ascending and descending order.

While sort only sorts in ascending order, gsort can be used for sorting in descending order. To sort in descending order, "-" must be used with a variable.

Example:

gsort sid –last_withdrawal_code - sorts by sid and then in descending order by last_withdrawal_code

bysort varlist: stata_cmd - Organizes the data by certain variables and executes a command.

Most Stata commands allow the bysort prefix, which repeats the command by each group of observations and sorts them under the variables listed.

Example:

```
sort sid school_year
bysort sid: gen count =_n
// once the data is sorted by sid and school_year, creates a counter for each observation by student.
```

sid	school_year	count
1	2005	1
1	2006	2
1	2007	3
2	2005	1
2	2006	2

order varlist | _all [, first | last | before() | after() | alpha] - Reorders selected variables in a dataset.

Orders the variables within the dataset, using the selected criteria one is specified. No sorting occurs, only the order of the variables is changed.

Example:

```
order sid school_year school_name
order sid, first
order school_year, before(school_name)
order _all, alpha
```

rename old_varname new_varname - Changes the name of an existing variable.

Example:

```
rename studentid sid
```

renvars [varlist] , transformation_option - Renames multiple variables.

Note: renvars is a package that you can install to Stata. The package can be installed using the following command: findit renvars and following the link to install it.

The most common transformation_options are:
upper, lower - converts the variable name to upper or lower case
prefix(str), postfix(str) - adds a specified string to the beginning or end of the variable name
substr(str1 str2) - substitutes (all occurrences of) str1 by str2 in variable names. str2 may be an empty string.

Example:

renvars can be useful after a reshape command, when your newly created variables have a postfix of a number based on a numeric value used in the reshape.

```
renvars *, subst(_1 _black)
renvars *, subst(_2 _hisp)
renvars *, subst(_3 _black)

old variable names:
score_1
score_2
score_3

new variable names:
score_black
score_hispanic
score_white
```

label - Labels variables or values of variables.

label variable varname "label" - labels the variable with the specified label. This label will show in the Stata Variable panel.

label define lblname # "label" [# "label" ...] [, add replace] - defines a label to be used for values of variables

label values varlist lblname labels the values of the variables included in varlist with the label specified. This label will be shown in the data editor and in tabulations of the variable.

label drop [varlist | _all] drops the specified or all labels.

Example:

```
label sid "Student ID"  // adds a variable label that is displayed in the variables window in Stata
label define genderlabel 1 "Male" 2 "Female"
label values gender genderlabel // adds value labels displayed in the browser, tabs, and lists
```

format varlist %fmt - Sets the output format for the variable.

The most typical formats include:

%#.#f	fixed numeric format
%#.#fc	fixed numeric format with comma separator
%td	date format

Example:

```
format -> display
format SAT_score %9.0f   1100
format SAT_Score %9.1fc  1,100.0
format school_end %td 30jun2008
```

list [varlist] [if] - Lists values of selected variables.

list displays the values of variables. If no varlist is specified, the values of all the variables are displayed.

Example:

```
sort school_name
list school_name if grade_span == "High" & school_name != school_name[_n-1]
// lists unique names of high schools
```

```
     +------------------------------+
     |                  school_name |
     |------------------------------|
 16. |            Camino High School |
 26. |           Central High School |
 41. |              City High School |
103. |             North High School |
117. |           Orchard High School |
128. |              Peak High School |
148. |             River High School |
154. |             South High School |
165. |            Upland High School |
171. | West Spring Hill High School |
177. |       White Sands High School |
     +------------------------------+
```

2.4 Using System Variables

_n - Contains the number of current observation.
_N - Contains the total number of observations in the database.

The _n and _N variables are useful for indexing observations or generating sequences of numbers. _n can act as a running counter within a by-group and _N acts as the total number within each by-group.

Example:

```
sort sid school_year
bysort sid: gen counter =_n // once the data is sorted by sid and school_year, creates a counter for each observation by student
bysort sid: gen total_count =_N // generates the total count of observations by student.
```

sid	school_year	count	total_count
1	2005	1	3
1	2006	2	3
1	2007	3	3
2	2005	1	2
2	2006	2	2

3 FUNCTIONS

3.1 String Functions

trim(s | varname) - Removes trailing spaces from a string.

This function can be used with any string, but typically is used with a variable name. It returns the string or variable without leading and trailing blanks. `trim("  this  ") = "this"`

Example:

```
replace lastname = trim(lastname)

lastname  ->   lastname
"  Smith  "    "Smith"
```

lower() - Returns a lower case version of a string.

Example:

```
replace lastname = lower(lastname)

lastname  ->   lastname
"smiTH "       "smith"
```

upper() - Returns an upper case version of a string.

Example:

```
replace lastname = upper(lastname)

lastname  ->   lastname
"smiTH "       "SMITH"
```

proper() - Returns a string with the first letter of every word capitalized.

Example:

```
replace lastname = proper(lastname)

lastname  ->   lastname
"smiTH"        "Smith"
```

regexm(s,re) - Performs a match of a regular expression and evaluates to 1 if regular expression re is satisfied by the string s, otherwise returns 0.

Example:

```
generate highschool = regexm(school_name, "High") - populates the variable highschool with 1 if the school name contains the string "High"; otherwise, it populates with 0.
```

school_name	highschool
Alpha Elementary School	0
Beta Middle School	0
White Sands High School	1

regexr(s1,re,s2) - Replaces the first substring within s1 that matches re with s2 and returns the resulting string.

Example:

```
generate school_name_short = regexr(school_name," School","")
replace school_name_short = regexr(school_name,"High","HS")
replace school_name_short = regexr(school_name,"Middle","MS")
replace school_name_short = regexr(school_name,"Elementary","ES")
```

school_name	school_name_short
Alpha Elementary School	Alpha ES
Beta Middle School	Beta MS
White Sands High School	White Sands HS

Regex functions can be used with regular expressions to perform complex string operations. A good description of regular expressions can be found at http://www.stata.com/support/faqs/data/regex.html

inlist(z, a,b,...) - returns 1 if z is a member of the remaining arguments; otherwise, it returns 0.

Example:

```
generate math_flag = 1 if inlist(course_code_desc, "ALGEBRA", "GEOMETRY")
```

course_code_desc	math_flag
ALGEBRA	1
ENGLISH	.
BIOLOGY	.
GEOMETRY	1

3.2 Math Functions

min(x1,x2,...,xn) - Returns the minimum value of a specified variable or number.

x1… xn can be a number, but most frequently it is used with variable names.

Example:

```
gen min_score = min(score_2006, score_2007)
```

sid	score_2006	score_2007	min_score
1	100	90	90
2	80	110	80

```
by sid: egen min_score = min(score)
```

sid	school_year	score	total_count
1	2005	100	90
1	2006	90	90
2	2005	80	80
2	2006	110	80

max (x1,x2,...,xn) - Returns the maximum value of a specified variable or number.

Use max just as you would min.

round (x[,y]) - Returns values rounded to the nearest integer (unless specified otherwise).

For `y = 1`, or with `y` omitted, Stata returns the closest integer to `x`;
With `y =.01`, for instance, `x` is rounded to two decimal places;
With `y = 5`, `x` is rounded to the closest multiple of 5.

Example:

```
gen round1 = round(score)
gen round2 = round(score, 0.1)
gen round3 = round(score, 5)
```

score	round1	round2	round3
28.25	28	28.3	30

3.3 Statistical Functions

center varlist, [standardize | generate(newvar)] - Standardizes a variable to have a mean of 0 and a standard deviation of 1.

Note: `center` is a package that you can install to Stata. The package can be installed using the following command: `ssc install center`.

For each `varname` in `varlist`, a new variable `c_varname` is created containing the centered (or standardized) values of that variable. If the `generate` option is used, you will be able to dictate the new variable's name.

Example:

```
center score, standardize
```

sid	score	c_score
1	85	0.70
1	75	-0.21
2	90	1.16
2	78	0.07

xtile newvar = exp [if] [,nquantiles(#)] - Creates 'quantile' categories for a variable.

`nquantiles(#)` defines the number of quantiles; default is 2.

Example:

```
xtile quartile = score, n(4)
```

sid	score	quartile
1	85	3
1	75	2
2	90	4
2	78	2

3.4 Date Functions

`date(s1,s2)` - returns the number of days since 01jan1960 corresponding to s1 based on s2. S1 contains the date recoded as a string in any format; s2 is any permutation of M, D and Y that describes s1.

Example:

```
date("20 march 1985", "DMY")   = 20mar1985        9210 days since 01jan1960
```

`mdy(M,D,Y)` - Returns the days since 01jan1960 after inputting (M,D,Y) where M is (1 12) D is (1 31) and Y is (0100 9999) (but probably 1800 to 2100).

Example:

```
mdy(3,20,1985)  = 20mar1985        9210 days since 01jan1960
```

`local lclname [=] ["]exp["]` - Assigns strings to local macro names.

Both double quotes (" and ") and compound double quotes (`" and "') are allowed. If the string has embedded quotes, compound double quotes are needed (see Quotes in the Introduction Chapter for more information).

Local macros are only kept in memory while Stata is running a selected set of commands. For example, if you run a command that assigns a local, then highlight another selection of your do file that references that local and run this selection, the local will not be recognized. The section that assigns the local and the section that references it have to be run together.

Local macros are referenced by enclosing them in single quotes. The left single quote is the character typically located at the top left of your keyboard, near the number 1. The left single quote is the standard single quote, or apostrophe.

Example:

```
local gend = 1
replace building = 1 if gender == `gend'
// Populates the variable building with the number 1 if the gender variable for that observation is
equal to the local macro gend,  that we set to be 1.

foreach var in asian black white {
     gen score_`var' = score if race_ethnicity == "`var'"
}
// Creates three new variables: score_asian, score_black and score_white, and populates these
variables with the value of the variable score if the race_ethnicity of that observation is equal to
the respective value of the local macro.
```

sid	race_ethnicity	score	score_asian	score_black	score_white
1	Asian-American	1000	1000		
2	Hispanic	1050			
3	African American	1100		1100	
4	White	1010			1010

`global mname [=] ["]exp["]` - Assigns strings to a global macro name (mname).

Global macros also assign temporary variables, but unlike locals, they are maintained in memory until the Stata window is closed.

Global macros are useful to assign at the beginning of your do file. You can use them for information that may change if you rerun the do file in the future. By assigning these macros as globals at the top of your do file you can avoid hard-coding values in your do file.

Global macros are referenced by preceding them by the `$` sign. You can also enclose the global in brackets; this practice is useful if the global macro is followed by string: eg. `$global_text`. In this case, Stata doesn't know where global ends and the text begins. Use instead `${global}_text`.

Example:

```
global currentyear "2007"
replace stillenrolled = 1 if school_year = $currentyear
```

5 OPERATORS

>, <, >=, <=, == - Operators that describe a variable as greater than, less than, greater than or equal to , less than or equal to, or equal to.

Example:

```
keep if grade_level >= 9
drop if total_days_enrolled == 0
```

!= or ~= - Indicates a variable is not equal to something else.

Example:

```
tab last_withdrawal_reason if sid != sid[_n-1]
```

&, | - Logical operators describing "and" and "or"

Example:

```
tab last_schy if last_wd_group == 3 & sid != sid[_n-1]
gen n_college_2yr = (yr2_yr4 == "2-year") | (yr2_yr4 == "Less Than 2 Years")
```

missing - Returns "1" if an argument is missing, returns a 0 if it is not missing.

`missing` has many uses:

It can serve as an option in the tabulation command to direct Stata to display missing values in addition to the populated values of the variable. It can also be a criteria for performing a certain command. `mi()` indicates that the command should be performed if the value of the variable is missing, while `!mi()` indicates that the command should be performed if the value of the variable is not missing.

`mi()` is equivalent to `==.` in the case of numeric values and `==""` in the case of string values.
`!mi()` is equivalent to `!=.` in the case of numeric values and `!=""` in the case of string values.

Example:

```
tab school_year, mi
// tabulates school_year, including missing values
replace gifted_ever = 0 if mi(gifted_ever)
// replaces missing values with 0
count if frpl_ever !=.
// counts observations where fprl_ever is not missing
```

6 COMMANDS

`assert exp` - Allows us to test if something is true about the data.

If the assertion is true, Stata continues to the next command.
If the assertion is false, Stata stops your program from continuing. This is useful when we want to know if we completed a task correctly.

Example:

```
assert highest_test_score != 0
// If none of the highest_test_score values are missing, Stata continues to the next command in the
do file.

assert highest_test_score > 88
// If there are observations where highest_test_score is less than or equal to 88, Stata stops and
displays an error message.
```

`isid varlist` - Checks if data is unique by a listed variable(s).

If the dataset is unique by the variable or combination of variables listed, Stata continues to the next command.
If the dataset is not unique by these variables, Stata stops and displays an error message. This is useful to test that our assumption about the uniqueness of the dataset is correct.

Example:

```
isid sid
variable sid does not uniquely identify the observations

isid sid school_year
```

display – Displays a message in the Stata window.

You can use display to display a message when Stata reached certain sections in your do file. It can be used in loops to display the current value of the variable. When used with variable names, Stata displays the value of the first record in the dataset.

Example:

```
display "Start cleaning SAT scores"
foreach subj in MA ELA {
     display "`subject'"
}

// Given the following dataset in memory:
```

sid	school_year
1	2005
1	2006

```
display school_year
2005
```

quietly – Suppresses output for the duration of the command.

If quietly is specified, Stata does not display any output of the command, but still returns the results of the operation.

Example:

```
quietly reshape long test_score, i(sid) j(school_year)
// Stata reshapes the dataset, but does not display any message on the screen or in the log file.
quietly count
// While the result of the count is not displayed, you can still use the scalar `r(N)' in commands
following count.
```

noisily – Ensures terminal output for the duration of the command.

If `noisily` is specified, Stata displays a message on the screen and in the log file. This is typical behavior in most cases, but it has to be specified with capture to see the message returned (see below).

capture – Executes command, suppressing all its output (including error messages, if any) and issues a return code of zero.

Preceding commands with the word `capture` allows the do-file or program to continue despite errors.

Example:

```
capture noisily isid sid
variable sid does not uniquely identify the observations
// The error message is displayed, but Stata continues. This can be useful for debugging.
```

7 LOOPS

foreach – Sets a local macro to each element in a list and executes commands enclosed in braces. The loop is executed the number of times the code.

Foreach can have multiple forms. The most common are:

```
foreach lname in any_list {}          any_list can be a list of strings or numbers
foreach lname of varlist  varlist {}     varlist is a list of variables in the current dataset
foreach lname of numlist  numlist {}     numlist is a list of numbers
```

Example:

```
foreach type in first last longest {
```

```
     gen `type'_hs_code = school_code
}
// Loops through the values  specified and generates three variables: first_hs_code, last_hs_code and
longest_hs_code, and populates it with the value of school_code.

foreach var of varlist frpl iep ell gifted {
     bys sid: egen `var'_ever = max(`var')
}
// Loops through the variables and executes a command for each. If your variables are in order, you
can also use foreach var of varlist frpl - gifted; This is only advised if you previously ordered
your variables, otherwise you may execute the command on unwanted variables, or miss some that you
intended to include.

foreach yr of numlist 1/4 {
     egen temp_credits_earned_yr`yr' = total(credits_earned) if year_in_hs <= `yr', by(sid)
}
```

forvalues - Sets a macro to each element of a numeric list and executes commands enclosed in curly braces.

forvalues loops through consecutive numeric values. It allows you to specify the steps you want to go through the numbers.

Example:

```
forvalues yr = 1/4 {
     generate status_after_yr`yr' = .
}
// Generates four variables: status_after_yr1 through status_after_yr4. This command is the same as
foreach yr of numlist 1/4 {}

forvalues i = 1(10)100 {
     di `i'
}
// Displays every 10th number from 1 to 100.
```

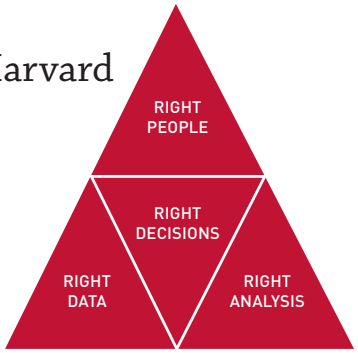
levelsof varname - displays a sorted list of the distinct values of a variable.

```
levelsof school_year, local(year)
// Creates a local variable containing all values of school year in the test score research file
foreach yr of local year {
     xtile qrt_8_`subject'_`yr' = test_`subject'_8 if school_year == `yr' , nq(4)
}
// Loops through the values of the school years, and creates variables that captures the quartile of
students' 8th grade tests within each school year.
```

The Strategic Data Project

OVERVIEW

The Strategic Data Project (SDP), housed at the Center for Education Policy Research at Harvard University, partners with school districts, school networks, and state agencies across the US. **Our mission is to transform the use of data in education to improve student achievement.** We believe that with the right people, the right data, and the right analyses, we can improve the quality of strategic policy and management decisions.



SDP AT A GLANCE

23 AGENCY PARTNERS
14 SCHOOL DISTRICTS
7 STATE EDUCATION DEPARTMENTS
2 CHARTER SCHOOL ORGANIZATIONS

79 FELLOWS
54 CURRENT
25 ALUMNI

CORE STRATEGIES

1. Placing and supporting top-notch analytic leaders as "Fellows"for two years with our partner agencies

2. Conducting rigorous diagnostic analyses of teacher effectiveness and college-going success using existing agency data

3. Disseminating our tools, methods, and lessons learned to many more education agencies

SDP DIAGNOSTICS

SDP's second core strategy, conducting rigorous diagnostic analyses using existing agency data, focuses on two core areas: (1) college-going success and attainment for students and (2) human capital (primarily examining teacher effectiveness).

The diagnostics are a set of analyses that frame actionable questions for education leaders. By asking questions such as, "How well do students transition to postsecondary education?" or "How successfully is an agency recruiting effective teachers?" we support education leaders to develop a deep understanding of student achievement in their agency.

ABOUT THE SDP TOOLKIT FOR EFFECTIVE DATA USE

SDP's third core strategy is to disseminate our tools, methods, and lessons learned to many more educational agencies. This toolkit is meant to help analysts in all educational agencies collect data and produce meaningful analyses in the areas of college-going success and teacher effectiveness. Notably, the analyses in this release of our toolkit primarily support questions related to college-going success. The data collection (Identify) and best practices (Adopt) stages of the toolkit, however, are applicable to any sort of diagnostic and convey general data use guidelines valuable to any analysts interested in increasing the quality and rigor of their analyses. Later releases will address analyses relating to teacher effectiveness.

Center for Education Policy Research
HARVARD UNIVERSITY

CENTER FOR EDUCATION POLICY RESEARCH
STRATEGIC DATA PROJECT
50 CHURCH ST., 4TH FLOOR, CAMBRIDGE, MA 02138
VOX 617.496.1563
FAX 617.495.2614
WWW.GSE.HARVARD.EDU/SDP