Stata Commands Useful for Importing, Inspecting, and Cleaning Data

1. Reading data from other formats – use `insheet`, `infile`, or `infix`. (I often find it easier to use File...Import from the drop-down menus. It is also possible to copy and paste data from Excel into the Stata Data Editor—but do this with caution. The latter is not easily replicated). `infile` can refer to a *data dictionary* (.dct) file that provides the necessary information to Stata for reading the data. Example for reading a CSV file:

```
insheet using "C:\data\myfilename.csv", comma
```

2. Search text in variable names and labels

```
lookfor keyword
```

3. Get quick details about variables (non-missing observations, unique values, mean, min, max, label) using `compact`, and identifying possible problem variables using `problems`

```
codebook, compact
codebook, problems
```

4. Getting a list of observations with missing values using `nmissing`. (This is a user-written ado file that may need to be installed first using `ssc install`; installation only needs to happen once per machine)

```
ssc install nmissing
nmissing
nmissing varlist
```

5. Confirm whether (or not) a variable is a unique identifier. The command `isid` will return nothing if the variable is unique, and will return an error message otherwise.

```
* note: idvar is suspected to be unique
isid idvar
```

6. Another way to see whether a variable is a unique identifier is by using `duplicates report`. This command has lots of other useful applications as well. The variable list can include multiple variables (e.g. *idvar* and *year*)

```
duplicates report idvar
```

7. Assigning user notes to a specific variable. You can include the characters TS to add a time-stamp. The command `notes:` by itself will add user notes to the dataset.

```
notes var1: Original values of 88 and 99 have been
     set to missing
notes var2: Original variable name was B1ETH
```

Last updated: August 30, 2021

```
* below: to view notes for var1
notes var1
notes list var1
```

8. Labeling *values* – requires that you define a set of label values first, and then apply them using the command `label values`

```
label define lblname # "label" # "label" ...
label values varname lblname
label dir
label list
label drop
```

9. Create an exact clone of an existing variable – this copies the variable's data, label, and value labels exactly. This is useful if you want to modify the variable and leave the original intact. Note, however, that the old value labels may no longer apply if the variable is later re-coded.

```
clonevar var1 = var2
```

10. Replacing instances of a variable that meets certain conditions. Be cognizant of how your command affects missing values (Stata treats missing values as large numbers).

```
replace var1 = 1 if var1 = 100
replace var2 = . if var2 < 0
```

11. Recoding – when you want to recode values of a variable that meet a certain condition. For example, you may want to re-categorize values (e.g. 1's to 1, 2's to 1, 3's to a 2, and 4's to a 2), or ranges of values (e.g. 1-10 set to 1, 11-15 set to 2, and so on).

```
* recoding specific values
recode varlist (1=1) (2=1) (3=2) (4=2)
recode var1 (1 2 3 4 5 = 1) (6 7 8 9 10 = 2)

* recoding ranges of values
recode varlist (1/10 = 1) (11/15 = 2)
```

Recode without the option `generate` will over-write the original variable. If you want to re-code into a *new* variable, use `generate`. For example:

```
recode var1 1=1 2=1 3=2 4=2, gen(newvar)
```

12. Encoding – when you want to assign numeric codes to unique values of a string variable. For example, suppose you have a string `state` that takes on four values: Connecticut, Massachusetts, New Jersey, and New York. The `encode` command will create a new numeric variable that takes on the values 1, 2, 3, and 4 for these (where the order is alphabetical). Variable labels will be created for the numeric codes.

```
encode state, gen(statenum)
```

13. Reviewing the contents of a dataset – use the Variables and Properties windows, the Variables manager (Data → Variables Manager), and/or these commands:

```
browse              (allows you to view the data like a spreadsheet)
edit                (allows you to edit the data like a spreadsheet—not advised)
list                (lists entire dataset in the Results window—not advised)
list in 1/10        (lists observations 1-10 only)
browse var1 var2    (view only the variables var1 and var2 like a spreadsheet)
list var1 var2      (lists only the variables var1 and var2)

describe            (basic information about variables—like a table of contents)
describe varnames   (basic information about specific variables)
desc, short         (example of abbreviated command name and option)

codebook varname      (to get a little more information about a variable)
inspect varname       (to get a little more information about a variable)
count                 (to see the number of observations)
count if condition    (to see the number of obs. where a condition is true)
```

14. Simple descriptive statistics, tables, and frequency distributions

```
summarize
sum varnames
sum varnames, detail

tab1 varname
tab1 varname, nolabel  (frequency distribution without value labels)
tab1 varname, missing  (frequency distribution including missing)

table varname
```

15. An alternative command, useful for obtaining descriptive statistics by subgroup. The option "stat" allows you to choose which statistics to display. Type `help tabstat` for a list of available statistics.

```
tabstat varlist, by(var1) stat(mean sd)
```

16. Histogram – use for numeric, continuous variables

```
histogram varname
```

17. Creating indicator (dummy) variables – examples

```
* Method 1: generate and replace
gen middle = 0
```

```
replace middle = 1 if grade==6 | grade==7 | grade==8

* Method 2: generate command with condition
gen middle==(grade==6 | grade==7 | grade==8)

* Method 3: create dummies for each unique categorical
     value. Here grdum is a prefix
tabulate grade, generate(grdum)
```

In all of the above cases, be aware of how missing values are treated. In some applications, you may be fine with the indicator variable (e.g. *middle*) being set to zero whenever values of the condition variable (e.g. *grade*) are missing. In other cases you may want the indicator variable to also be missing.

18. Extended generate (perhaps my favorite command) – is particularly useful when you need to create a variable that summarizes data within groups (e.g. a within-group mean)

```
* Examples
egen newvar = mean(var1), by(var2)

* Creating a z-score for mathscore:
egen temp1 = mean(mathscore)
egen temp2 = sd(mathscore)
gen zmath = (mathscore - temp1) / temp2

or just:
egen zmath2 = std(mathscore)
```

Other egen functions include, among many others: count (counting nonmissing observations by group), cut (creating equal sized groups), group (creating indicators for membership within a certain group defined by *varlist*), max, min, mean, median, sd, mode, pctile (percentile), rank, etc.

19. Converting string to numeric and vice versa (use with caution if over-writing the original)

```
destring oldvar , force replace
destring oldvar , force gen(newvar)
tostring oldvar , force replace
tostring oldvar , force gen(newvar)
```

20. Exporting data to another format, such as comma-separated values (CSV). (I often find it easier here to use File...Export).

```
outsheet using "C:\data\myfilename.csv", comma
```