# Introduction to Social Data Analytics

Class 8

## Today: Continue with Stata

By the end of today's lecture, you should be able to:

- Identify variable types and recall best practices when creating variables
- Assign values to variables using functions and logical operators/statements
- Sort data and assign values to variables by group designation

## Today's Structure

- Load titanic.dta if you haven't already
- Introduce new Stata commands and practice using them in the Command window
- Work in pairs to finish class8.do that you started for the pre class exercise

## String vs numeric variables

Last class we learned how to generate a numeric variable. Let's try generating a string variable:

- `gen crew = "no"`
- `replace crew = "yes" if class == 0`

## String vs numeric variables

Last class we learned how to generate a numeric variable. Let's try generating a string variable:

- `gen crew = "no"`
- `replace crew = "yes" if class == 0`

Text strings must *always* be bounded by quotes.

## String vs numeric variables

Last class we learned how to generate a numeric variable. Let's try generating a string variable:

- `gen crew = "no"`
- `replace crew = "yes" if class == 0`

Text strings must *always* be bounded by quotes.

Use `desc` to check the storage type of each variable.

## Some notes on naming variables

- No spaces allowed, case sensitive
- Up to 32 characters
- Rules of thumb

## Some notes on naming variables

- No spaces allowed, case sensitive
- Up to 32 characters
- Rules of thumb
    - Keep it short, e.g. *educ, income*
    - Make it informative: e.g. *female* instead of *gender*
    - Maintain consistency, e.g. *ln_income*, *ln_wage*, *ln_tax*

## Command: egen

egen is like gen, but it's used for assigning values to variables with functions that work across all observations. Try:

```
gen mean_survive = mean(survive)
```

## Command: egen

egen is like gen, but it's used for assigning values to variables with functions that work across all observations. Try:

```
gen mean_survive = mean(survive)
```

The above fails to work. You must use egen:

```
egen mean_survive = mean(survive)
```

## Generating variables by group

Suppose we want to calculate the mean survival rate by class. We can accomplish this using `sort`, `by`, and `egen`:

- `sort class id`
- `by class:  egen mean_survive_class = mean(survive)`

## Generating variables by group

Suppose we want to calculate the mean survival rate by class. We can accomplish this using `sort`, `by`, and `egen`:

- `sort class id`
- `by class:  egen mean_survive_class = mean(survive)`

`sort` orders the data from smallest to largest numerically or alphabetically (if string). If multiple variables are listed, the data are sorted lexicographically.

## Generating variables by group

Suppose we want to calculate the mean survival rate by class. We can accomplish this using `sort`, `by`, and `egen`:

- `sort class id`
- `by class:  egen mean_survive_class = mean(survive)`

`sort` orders the data from smallest to largest numerically or alphabetically (if string). If multiple variables are listed, the data are sorted lexicographically.

`by class` tells the program to take the mean within each class separately. The data must be sorted to apply commands within designated groups.

## Commands: `if`, `in`, & `list`

- `if` restricts the command to certain criteria, *for each observation*, similar to IF in Excel

## Commands: `if`, `in`, & `list`

- `if` restricts the command to certain criteria, *for each observation*, similar to IF in Excel
- `in` restricts the observations to which a function applies.

## Commands: `if`, `in`, & `list`

- `if` restricts the command to certain criteria, *for each observation*, similar to IF in Excel
- `in` restricts the observations to which a function applies.
- `list` displays observations for specified variables or all variables if unspecified.

## Commands: `if`, `in`, & `list`

- `if` restricts the command to certain criteria, *for each observation*, similar to IF in Excel
- `in` restricts the observations to which a function applies.
- `list` displays observations for specified variables or all variables if unspecified.

Try it:

```
list in 1/200 if male == 0
list id survive in 1/100 if male == 1
```

## Commands: `keep` & `drop`

Both of these commands are used to keep or drop (delete) observations or variables. Try the following:

- `keep in 1/2000`
- `drop in 1001/2000`
- `keep id adult survive`
- `drop adult`

## Commands: `keep` & `drop`

Both of these commands are used to keep or drop (delete) observations or variables. Try the following:

- `keep in 1/2000`
- `drop in 1001/2000`
- `keep id adult survive`
- `drop adult`

Remember: these changes are made to the data in working memory. The original dataset (titanic.dta) remains unchanged unless you save it.

## Commands: `keep` & `drop`

Both of these commands are used to keep or drop (delete) observations or variables. Try the following:

- `keep in 1/2000`
- `drop in 1001/2000`
- `keep id adult survive`
- `drop adult`

Remember: these changes are made to the data in working memory. The original dataset (titanic.dta) remains unchanged unless you save it.

Go ahead and restore the data with: `use titanic, clear`

## Logic in Stata

- `&` means ‘AND’ ; | means ‘OR’ ; `!` means ‘NOT’
- Use parenthesis when necessary. In Stata, `&` takes precedence over |

## Logic in Stata

- `&` means 'AND' ; | means 'OR' ; `!` means 'NOT'
- Use parenthesis when necessary. In Stata, `&` takes precedence over |

`a | b & c` is **not** the same as `(a | b) & c`

## Logic in Stata

- `&` means 'AND' ; `|` means 'OR' ; `!` means 'NOT'
- Use parenthesis when necessary. In Stata, `&` takes precedence over `|`

`a | b & c` is **not** the same as `(a | b) & c`

`a | b & c` **is** the same as `a | (b & c)`

## Logic in Stata

- & means 'AND' ; | means 'OR' ; `!` means 'NOT'
- Use parenthesis when necessary. In Stata, & takes precedence over |

`a | b & c` is **not** the same as `(a | b) & c`

`a | b & c` **is** the same as `a | (b & c)`

Remember, use double equals signs to perform a logical test,
e.g. `list if id == 100`

## Logic in Stata

- & means 'AND' ; | means 'OR' ; `!` means 'NOT'
- Use parenthesis when necessary. In Stata, & takes precedence over |

`a | b & c` is **not** the same as `(a | b) & c`

`a | b & c` **is** the same as `a | (b & c)`

Remember, use double equals signs to perform a logical test,
e.g. `list if id == 100`

Use single equals signs to assign values, e.g. `gen var = 100`

## Time to work on class8.do

Here are the commands/operators we covered today:

- `egen`
- `if`
- `in`
- `list`
- `keep`
- `drop`
- `&`, `|`, `!`
- = vs ==