**References**

[**https://data.princeton.edu/stata/**](https://data.princeton.edu/stata/)

[**https://stats.idre.ucla.edu/stata/seminars/notes15/modifying15/**](https://stats.idre.ucla.edu/stata/seminars/notes15/modifying15/)

<http://www.danielmsullivan.com/pages/tutorial_stata_to_python.html>

<http://pandas.pydata.org/pandas-docs/stable/getting_started/comparison/comparison_with_stata.html>

**Reading and Saving Files**

**Reading .dta(stata) file**

|  |  |
| --- | --- |
| Stata | use <varlist> using <dtafile> |
| Python | pd.read\_stata('<dtafile>', columns=<varlist>) |

**Using .dta file with only specific columns**

|  |  |
| --- | --- |
| Stata | use <varlist> using <dtafile> |
| Python | pd.read\_stata('<dtafile>', columns=<varlist>) |

**Read in .xlsx or .csv files**

|  |  |
| --- | --- |
| Stata | import excel using <excelfile> |
| Python | pd.read\_excel('<excelfile>') in python |

|  |  |
| --- | --- |
| Stata | import delimited using <csvfile> |
| Python | pd.read\_csv('<csvfile>') in python |

**Save as .dta file to certain directory**

|  |  |
| --- | --- |
| Stata | save <filename>, replace |
| Python | df.to\_stata('<filename>') OR  df.to\_pickle('<filename>') (for Python-native file type) |

**Save as .csv or excel file to certain directory**

|  |  |
| --- | --- |
| Stata | outsheet using <csv\_name>, comma |
| Python | df.to\_csv('<csv\_name>') |

|  |  |
| --- | --- |
| Stata | export excel using <excel\_name> |
| Python | df.to\_excel('<excel\_name>') |

use clear, replace

**Description and Summary Statistics**

**Breakdown of values and their counts**

|  |  |
| --- | --- |
| Stata | Tab var 🡺 count, percentage and cumulative percentage |
| Python | df[‘var’].value\_counts( ) 🡺 just count breakdown |

|  |  |
| --- | --- |
| Stata | describe |
| Python | df.info( ) OR df[<var>].dtype |

|  |  |
| --- | --- |
| Stata | count |
| Python | df.shape[0] OR len(df). Here df.shape returns a tuple with the length and width of the DataFrame. |

Cf. count if <condition> e.g. count if sex == 1

|  |  |
| --- | --- |
| Stata | summ var  summ var if <condition> |
| Python | df['var'].describe( )  df[<condition>][‘var’].describe( ) OR df.loc[condition, ‘var’].describe() |

|  |  |
| --- | --- |
| Stata | summ ‘var’ [aw = <weight>] |
| Python | Right now you have to calculate weighted summary stats manually. There are also some tools available in the Statsmodels package. |

|  |  |
| --- | --- |
| Stata | summ <var>, d |
| Python | df[<var>].describe() plus df[‘var’].quantile([.1, .25, .5, .75, .9]) or whatever other statistics you want. |

**Basic Data Wrangling and Operations (e.g. creation or dropping of new variable)**

**Sort observations using var or multiple vars**

|  |  |
| --- | --- |
| Stata | sort var1 var2… |
| Python | df.sort\_values([‘sorting variable name’]) |

**Rename Variable Names**

|  |  |
| --- | --- |
| Stata | rename var1 var2 🡺 rename var1 to var2  rename (var1 var3) (var2 var4) 🡺 rename var1 to var3 and rename var3 to var4 |
| Python | df.rename(columns = {‘var1’: ‘var2’}, inplace=True) |

**Keep or drop observations based on conditions**

|  |  |
| --- | --- |
| Stata | keep if <condition>  drop if <condition> |
| Python | df = df[<condition>]  df = df[~(<condition>)] |

**Keep or drop variables**

|  |  |
| --- | --- |
| Stata | keep var1 var2…  drop var1 var2… |
| Python | df = df[[‘var1’, ‘var2’]]  del df[‘var1’] OR df = df.drop(labels= [‘var1’,’var2’], axis=1) |

keep varstem\* df = df.filter(like='varstem\*')

drop varstem\* df = df.drop(df.filter(like='varstem\*').columns, axis=1)

**Create new variables and replace**

|  |  |
| --- | --- |
| Stata | gen newvar = <expression>  gen newvar = <expression> if <condition>  replace newvar = <expression> if <condition> |
| Python | df[‘newvar’] = <expression> |

|  |  |
| --- | --- |
| Stata | egen newvar = max(var)  egen newvar = mean(var) |
| Python | newvar = df[‘var’].max()  newvar = df[‘var’].mean() |

NOTE: For these egen commands, <newvar> is a full (constant) column in Stata, while it is a scalar in Python.

egen <newvar> = count(<var>)

🡺 <newvar> = df[<var>].notnull().sum().

egen <newvar> = group(<varlist>)

🡺 <newvar> = econtools.group\_id(df, cols=<varlist>)

egen <newvar> = total(<var>)

🡺 <newvar> = df[<var>].sum()

**Other Operations**

**Checking if certain values are included in a variable**

|  |  |
| --- | --- |
| Stata | inlist(var, “val1”, “val2”) |
| Python | df[‘var’].isin([“val1”, “val2”]) |

**Checking if values in a column/variable are within a specific range**

|  |  |
| --- | --- |
| Stata | inrange(<var>, ‘val1’, ‘val2’) |
| Python | df[‘var’].between(‘val1’, ‘val2’) \*\*inclusive = False / True |

**Setting Global directory**

global data “O:/kaggle/data”

use “${data}/titanic.csv”

**Preserve & Restore Operations**

|  |  |
| --- | --- |
| Stata | preserve  keep if missing(dob);  drop dob;  save “mystata/dob\_missing.dta”, replace;  restore |
| Python | Assign modified/wrangled dataframe in another variable |

**Certain utility functions not working … then**

global codedir "D:/Users/{uniqname}/Desktop/code”

**similar to try except**

capture

\_rc != 0 {

}

**When excel file is to big to import**

set excelxlsxlargefile on

**Displaying / logging output**

log using <file>

**String Operations**

**Split operations**

|  |  |
| --- | --- |
| Stata | split n, p(“,“) gen(n\_) 🡺 split the values in variable “n” on comma and make them into separate new columns named n\_1, n\_2… |
| Python | Df[‘n’].str.split( pat= “,”) |

**Removing Blanks**

|  |  |
| --- | --- |
| Stata | Say s is a string  stritrim: remove blanks  stritrim(s) returns s with all consecutive, internal blanks collapsed to one blank  strltrim(s) returns s with leading blanks removed  strrtrim(s) returns s with trailing blanks removed  strtrim(s) returns s with leading and trailing blanks removed.  e.g. replace name\_last = strtrim(stritrim(lower(name\_last))) |
| Python | df['last\_name'] = df['last\_name'].str.strip( )  OR df['last\_name'] = df['last\_name'].str.replace(" ","") |

**Regular expression**

|  |  |
| --- | --- |
| Stata | gen suffix = if regexm(n\_2, “[ ] (junior|senior|jr|sr|i|ii|iii|iv|v|v[i]+)$”); |
| Python |  |

**Parsing subset of strings**

|  |  |
| --- | --- |
| Stata | gen dob\_yr = substr(DOB, -4, . ) # parse 4th from last to last string  substr(raw\_name, 1, 3) #parse 3 characters starting from 1st indexed character from values in raw\_name column |
| Python | df[‘dob\_yr’] = df.DOB.apply(lambda x: x[-4: ]) |

**Substituting strings with some other value/string**

|  |  |
| --- | --- |
| Stata | subinstr(var, “x”, “”, . ) #replace “x” with empty string(“”) for all values in column var |
| Python | df[‘var’] = df.var.replace({“x”: “”}) |

**remove (or more like change them to empty string) if a value does not start with “$” (usually for amount vars)**

replace rest\_amt = “” if regex(rest\_amt, “(^[\$])”)

**remove (or more like change them to empty string) if a value does not have xx/xx/xx date format (usually for date vars)**

replace sent\_dt = “” if !regexm(sent\_dt, “(^[0-9]+/[0-9]+/[0-9]+$)”)

**Getting position index of a certain character in string**

strops(string, “\_\_”) : get position index of first occurrence of “\_\_” in string

strrpos(string, “\_\_”) : get position index of last occurrence of “\_\_” in string

**Getting unique number of observations (in this example, unique # of ids)**

egen x = group(id)

sum x

>> Variable | Obs Mean Std. Dev. Min Max

-------------+--------------------------------------------------------

x | 5 1.4 .5477226 1 2

The value for Max is the # of unique ids.

OR

tag = tag(var1 var2….)

ta tag 🡺 0: duplicates, 1: first unique

OR

by var, sort: gen nvals = \_n == 1

replace nvals = sum(nvals)

replace nvals = nvals[\_N]

*define distinct observations of a variable with respect to one or more other variables. With one more variable (say, foreign), we might want to calculate the number of* ***distinct observations of rep78 separately for each value of foreign***

by foreign rep78, sort: gen nvals = \_n == 1

by foreign: replace nvals = sum(nvals)

by foreign: replace nvals = nvals[\_N]

**Groupby Operations / Reshaping based on aggregate statistics**

Bys cjars\_id INCRNO (FR\_DATE TO\_DATE): gen inc\_entry\_dt = FR\_DATE if FR\_MOVE\_TYPE == “I” & [\_n] == 1;

= = = = = = = = = = = = = = = = =

*\*\*\*Fill in missing values using values from the same group*

bysort id (number): replace number=number[\_N]

That would sort the highest value observed in each block (which could be missing) to the end of each block and use that to overwrite all values in the same block, regardless of whether values were missing.

bysort id: replace number=number[1]

bysort id : replace number = number[\_n-1] if missing(number) & \_n > 1

= = = = = = == = = = = = = == = = = = =

**Reshaping data**

collapse (sd) <var> (median) <var> ///

(max) <var> (min) <var>, ///

by(<groupvars>)

🡺 df.groupby(<groupvars>)[<var>].agg(['std', 'median', 'min', 'max', 'sum'])

collapse (<stat>) <var> [iw = <weight>]

collapse (<stat>) <stat\_vars>, by(<groupvars>)

🡺 df.groupby(<groupvars>)[<stat\_vars>].<stat>()

recode <var> (1/5 = 1)

recode <var> (1/5 = 1), gen(<newvar>)

label var <var> <label>

label define <labelname> 1 <valuelabel>

label values <var> <labelname>

label list <labelname>

**===========================**

|  |  |
| --- | --- |
| reshape <wide/long> <stubs>, i(<vars>) j(<var>) |||||| | wide: df.unstack(<level>)  long: df.stack(<column\_level>)  see also df.pivot |

**Merge and Join Operations**

use “[PATH]”, clear;

merge 1:1 CASE\_NUMBER record\_id using [file name (without .dta)]

|  |  |  |
| --- | --- | --- |
| append using <filename> | | df\_joint = df1.append(df2) |
| merge 1:1 <vars> using <filename> | | df\_joint = df1.join(df2) if <vars> are the DataFrames' indexes, or  df\_joint = pd.merge(df1, df2, on=<vars>) otherwise. Beware  that pd.merge will not keep the index of either DataFrame. |
| *Pandas how* | *Stata , keep()* | *Intuition* |
| how='left' | keep(1, 3) | Keeps all observations in the "left" DataFrame. |
| how='right' | keep(2, 3) | Keeps all observations in the "right" DataFrame. |
| how='inner' | keep(3) | Keeps observations that are in both DataFrames. |
| how='outer' | keep(1 2 3) | Keeps all observations. |

**Some Date/Time Operations**

**Extracting year,month,day info from date-like formatted string**

year(date(adj\_sent\_dt, “MDY”));

month

day

<https://www.ssc.wisc.edu/sscc/pubs/stata_dates.htm>

**Changing DOB or some date variable that has mm/dd/yyyy 00:00:00 format with DOUBLE/FLOAT data type to string**

String(DOB, “%td”)

//

Gen newvar = String(DOB, “%tcDDmonCCYY”)

//

Replace inc\_entry\_dt = dofc(inc\_entry\_dt);

Replace inc\_exit\_dt = dofc(inc\_exit\_dt);

Format inc\_\*dt %td;

//

gen DOB = daily(string(DATE\_OF\_BIRTH, “%9.0f”), “YMD”)

format DOB %td

replace dob\_mm = month(date(string(DOB, “%td”), “DMY”))

**Setting index of panel data**

tsset <panelvar> <timevar>

🡺 df = df.set\_index([<panelvar>, <timevar>])

**Lags for time series**

L.<var>

🡺 df.shift() NOTE: The index must be correctly sorted for shift to work the way you want it to.

L2.<var>

🡺 df.shift(2)

F.<var>

🡺 df.shift(-1)

**Econometrics**

| Stata | Python |
| --- | --- |
| ttest <var>, by(<var>) | from scipy.stats import ttest\_ind  ttest\_ind(<array1>, <array2>) |
| xi: i.<var> | pd.get\_dummies(df[<var>]) |
| i.<var2>#c.<var1> | pd.get\_dummies(df[<var2>]).multiply(df[<var1>]) |
| reg <yvar> <xvar> if <condition>, r | import econtools.metrics as mt  results = mt.reg(df[<condition>], <yvar>, <xvar>, robust=True) |
| reg <yvar> <xvar> if <condition>, vce(cluster <clustervar>) | results = mt.reg(df[<condition>], <yvar>, <xvar>, cluster=<clustervar>) |
| areg <yvar> <xvar>, absorb(<fe\_var>) | results = mt.reg(df, <yvar>, <xvar>, a\_name=<fe\_var>) |
| predict <newvar>, resid | <newvar> = results.resid |
| predict <newvar>, xb | <newvar> = results.yhat |
| \_b[<var>], \_se[<var>] | results.beta[<var>], results.se[<var>] |
| test <varlist> | results.Ftest(<varlist>) |
| test <varlist>, equal | results.Ftest(<varlist>, equal=True) |
| lincom <var1> + <var2> | econtools.metrics.f\_test with appropriate parameters. |
| ivreg2 | econtools.metrics.ivreg |
| outreg2 | econtools.outreg |
| reghdfe | None (hoping to add it to Econtools soon). |

**Plotting / Visualization**

| *Stata* | *Python* |
| --- | --- |
| binscatter | econtools.binscatter |
| maptile | No quick tool, but easy to do with Cartopy. |
| coefplot | ax.scatter(results.beta.index, results.beta) often works. Depends on context. |
| twoway scatter <var1> <var2> | df.scatter(<var2>, <var1>) |
| twoway scatter <var1> <var2> if <condition> | df[<condition>].scatter(<var2>, <var1>) |
| twoway <connected/line/area/bar/rarea> | As above, though ax.plot(<var1>, <var2>) is better. Like merge, it's a different paradigm, needs more explanation. |

Move var1 variable right next to var2

order var1, after(var 2)