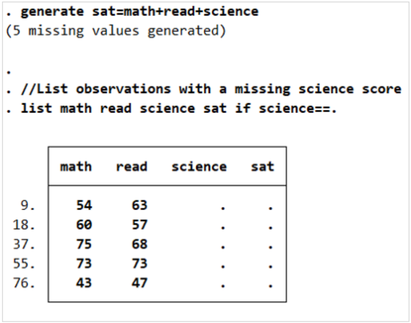
Data wrangling

Loading...

```
use "https://stats.idre.ucla.edu/stat/data/hs0", clear
```

Generating variables

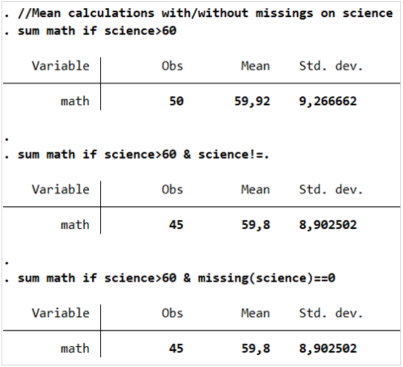
- Variables do not always arrive in the form that we need.
- Use **generate** to create new variables.
- You can use different operations on existing (numeric) variables to generate sums, differences, products, logarithms, squares, ...
- Note that if an input value to a generated variable is missing, the result will be missing as well.



```
. generate sat=math+read+science
(5 missing values generated)

.
. //List observations with a missing science score
. list math read science sat if science==.

     +------------------------------+
     | math   read   science    sat |
     |------------------------------|
  9. |   54     63         .      . |
 18. |   60     57         .      . |
 37. |   75     68         .      . |
 55. |   73     73         .      . |
 76. |   43     47         .      . |
     +------------------------------+
```

Missing values in Stata

- **Missing** `numerics` are represented by **.** (dot)
- **Missing** `strings` are represented by `""` (empty quotes)
- You can check for missing by testing for equality to . (or "" for string variables) or you can also use the `missing()` function.
- In model estimation, Stata drops missing observations from the analysis.
- Stata treats missings as if they were infinitely large numbers. When you want to calculate the mean math score of all students of science scores above 60, you have to exclude the students with missing science scores! Otherwise, you might get a wrong estimate for the mean.

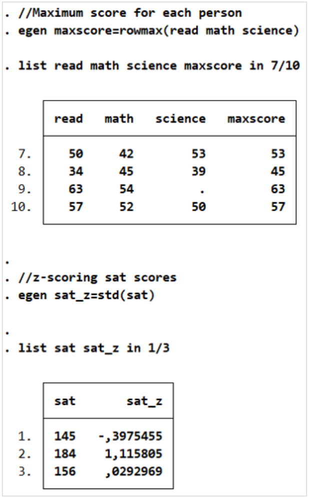


```
. //Mean calculations with/without missings on science
. sum math if science>60

    Variable |        Obs        Mean    Std. dev.
-------------+---------------------------------
        math |         50       59,92    9,266662

.
. sum math if science>60 & science!=.

    Variable |        Obs        Mean    Std. dev.
-------------+---------------------------------
        math |         45        59,8    8,902502

.
. sum math if science>60 & missing(science)==0

    Variable |        Obs        Mean    Std. dev.
-------------+---------------------------------
        math |         45        59,8    8,902502
```

Next generation variables

- **egen** (extended generate) creates variables using a wide array of more complicated functions.
- Statistical functions over multiple variables, e.g., means across several variables
- Functions that accept a single variable, but do not involve simple arithmetic operations, e.g., z-standardizing a variable, subgroup means

  help egen



```
. //Maximum score for each person
. egen maxscore=rowmax(read math science)

. list read math science maxscore in 7/10

     +-------------------------------------+
     | read   math   science   maxscore    |
     |-------------------------------------|
  7. |   50     42        53         53    |
  8. |   34     45        39         45    |
  9. |   63     54         .         63    |
 10. |   57     52        50         57    |
     +-------------------------------------+

.
. //z-scoring sat scores
. egen sat_z=std(sat)

.
. list sat sat_z in 1/3

     +------------------+
     | sat       sat_z  |
     |------------------|
  1. | 145   -,3975455  |
  2. | 184    1,115805  |
  3. | 156    ,0292969  |
     +------------------+
```

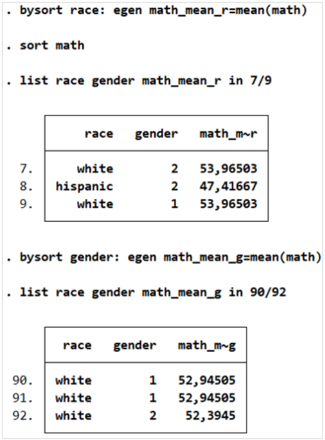
The prefix **by**

- Remember the *prefix* from our Stata syntax?

  [*prefix*:] `command` [*varlist*] [, *opt*]
- `by` [*varlist*] tells Stata to subset your data before executing the command.
- Stata performs the operation within groups specified by each unique value of *varlist*.

! Your data needs to be sorted. `bysort` does the trick. Variables in braces are used for sorting, but not for grouping the data.

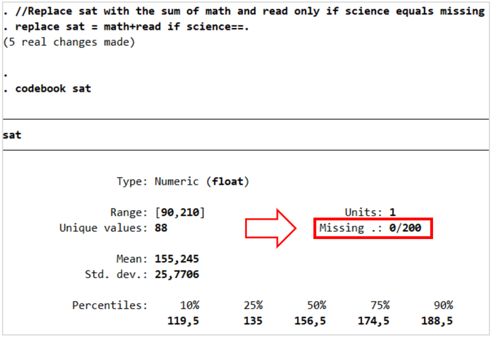
- Let's calculate mean math scores by ethnic background with our `egen` function.



```
. bysort race: egen math_mean_r=mean(math)

. sort math

. list race gender math_mean_r in 7/9

     +-------------------------------+
     |     race   gender   math_m~r  |
     |-------------------------------|
  7. |    white        2   53,96503  |
  8. | hispanic        2   47,41667  |
  9. |    white        1   53,96503  |
     +-------------------------------+

. bysort gender: egen math_mean_g=mean(math)

. list race gender math_mean_g in 90/92

     +----------------------------+
     |  race   gender   math_m~g  |
     |----------------------------|
 90. | white        1   52,94505  |
 91. | white        1   52,94505  |
 92. | white        2    52,3945  |
     +----------------------------+
```

Replacing values

- You can only **generate** variables once. So how do you manipulate existing variables?
- Use **replace** to change values of existing variables.
- Replace is often used with **if** to change values for a subset of observations.



```
. //Replace sat with the sum of math and read only if science equals missing
. replace sat = math+read if science==.
(5 real changes made)

.
. codebook sat

----------------------------------------------------------------------------
sat
----------------------------------------------------------------------------

                  Type: Numeric (float)

                 Range: [90,210]                    Units: 1
         Unique values: 88                      Missing .: 0/200

                  Mean: 155,245
             Std. dev.: 25,7706

           Percentiles:     10%       25%       50%       75%       90%
                          119,5       135     156,5     174,5     188,5
```

Changing it up

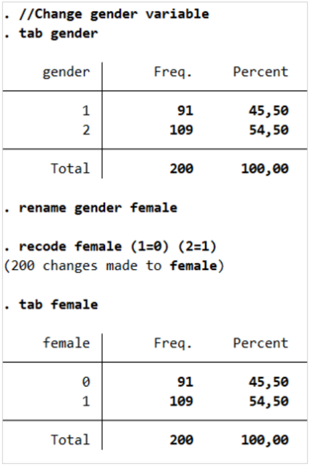
- If you want to change the name of a newly generated or an existing variable, use **rename**:

  rename *oldvar newvar*

- You can also make an exact copy of an existing variable into a new variable with **clonevar**:

  clonevar *newvar* = *oldvar*

- You often want to change variable values, i.e., when creating dummy variables. Use **recode** for this.
- Alternative: **replace** with an appropriate **if**-condition



```
. //Change gender variable
. tab gender

     gender |      Freq.     Percent
------------+-----------------------
          1 |         91       45,50
          2 |        109       54,50
------------+-----------------------
      Total |        200      100,00

. rename gender female

. recode female (1=0) (2=1)
(200 changes made to female)

. tab female

     female |      Freq.     Percent
------------+-----------------------
          0 |         91       45,50
          1 |        109       54,50
------------+-----------------------
      Total |        200      100,00
```

Giving your variables meaning

- You can use **variable labels** to let others know what a variable contains:

  `label` *varname* ["varlabel"]

- Use **value labels** to give meaning to values of `numeric` variables:

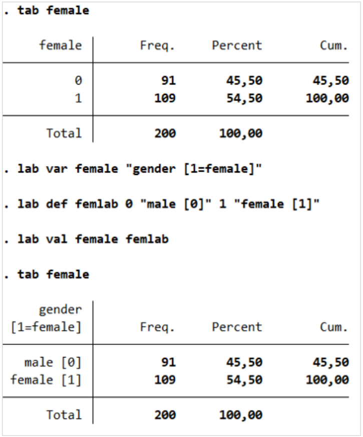
1: Define the value label:

   `label define` [*labname*] # ["vallabel [#]"]

2: Assign label to variable:

   `label values` [*varname*] [*labname*]

- Use of meta data as good coding practice!

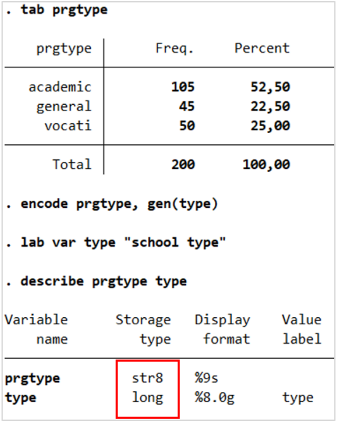


```
. tab female

     female |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |         91       45,50       45,50
          1 |        109       54,50      100,00
------------+-----------------------------------
      Total |        200      100,00

. lab var female "gender [1=female]"

. lab def femlab 0 "male [0]" 1 "female [1]"

. lab val female femlab

. tab female

     gender |
 [1=female] |      Freq.     Percent        Cum.
------------+-----------------------------------
   male [0] |         91       45,50       45,50
 female [1] |        109       54,50      100,00
------------+-----------------------------------
      Total |        200      100,00
```

Encoding strings into numerics

- **encode** converts a string variable into a numeric variable.
- Remember that some Stata commands require numeric variables!
- **encode** will use alphabetical order to order the numeric codes.
- **encode** will convert the original string values into a set of value labels.
- **encode** will create a new numeric variable *varname*, which must be specified as an option:

  , gen(*varname*)

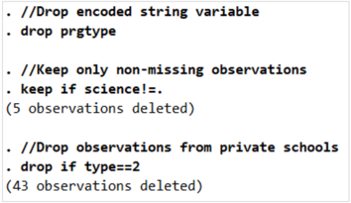


```
. tab prgtype

    prgtype |      Freq.     Percent
------------+-----------------------
   academic |        105       52,50
    general |         45       22,50
     vocati |         50       25,00
------------+-----------------------
      Total |        200      100,00

. encode prgtype, gen(type)

. lab var type "school type"

. describe prgtype type

Variable      Storage   Display    Value
    name         type    format    label
------------------------------------------
prgtype         str8    %9s
type            long    %8.0g      type
```

Keep it or drop it?

- In large datasets, it helps to only include relevant variables or observations to increase computation speed.
- `drop` deletes variables or observations and keeps the rest.
- `keep` preserves selected variables or observations and drops the rest.
- You can combine `keep` and `drop` with **if**-conditions when referring to observations

! Does not work with variables



```
. //Drop encoded string variable
. drop prgtype

. //Keep only non-missing observations
. keep if science!=.
(5 observations deleted)

. //Drop observations from private schools
. drop if type==2
(43 observations deleted)
```

Working with panel data

- In a panel, entities are observed more than once, e.g., annual household data, monthly country data, ...
- Use `xtset` to tell Stata that it currently stores panel data
  - The first identifier declares the unit of analysis, e.g., individuals in individual-level survey data.
  - The second identifier determines the time dimension of the data, e.g., years in annual surveys.
- Use `xtsum` and `xtdescribe` to investigate your panel.
- Working within panel units:
  - Use `bysort`ed variables with `[_n]` to access the n^{th} observation within a panel unit.
  - Use `[_n+1]` to access the next observation, `[_n-1]` for the previous observation.
  - Use `f.` and `l.` with a variable to access lags and leads while respecting the time dimension of the data.