

## Lesson 2: Basic Commands

This lesson introduces some basic commands, including listing variables, displaying summary information about variables, generating, replacing, and renaming variables, labelling variables, applying labels to numeric data, and removing data.

First, let's load in the **Lesson\_01\_{initials}.dta** file (the one with your initials):

```
use "Lesson_01_SH.dta", clear
```

Alternatively, load in the **Lesson\_02.dta** if you want a fresh dataset.

### 2.1 List

One of the simplest commands simply presents observations in a dataset. To do this, simply type:


**list**

1.	varl 1	age 78	sex Female	height 166	weight 134.4818	hair_c-r Black	systol-p 118	diasto-p 81	accomm-n Own	exercise 0	calories 1605	income <18,000	varl3 .	curren-r 2
	favourite_colours Blue; Yellow; Red					date-ement 03/12/2018			dat-tment 23nov2018		_merge matched (3)			
2.	varl 2	age 39	sex Female	height 167	weight 171.0785	hair_c-r Red	systol-p 134	diasto-p 96	accomm-n Rent	exercise 4	calories 2329	income <18,000	varl3 0	curren-r 2
	favourite_colours Red; Indigo; Violet					date-ement 31/12/2018			dat-tment 29dec2018		_merge matched (3)			

If the output goes on for a while, I'll only show a bit of it after each command to save space. Just typing **list** by itself will list all observations in all variables; if there isn't enough space on your screen to show all the variables on a single line, Stata will split each observation over multiple lines. It can help to restrict which variables are shown by typing them after **list**, for example if you only wanted to **list** *age* and *sex*:

```
list age sex
```

	age	sex
1.	78	Female
2.	39	Female
3.	79	male
4.	27	Female
5.	74	male

There' isn't much more to say about **list**, but depending on how your version of Stata is set up, Stata may wait for you to click [–more–](#) or press the spacebar once the results window is full. Alternatively, you can stop Stata from listing data by clicking the red cross  at the top of the screen. If you want to have Stata print out all the results without having to click [–more–](#)

, then type into the command bar:

```
set more off
```

This will force Stata to give you all the results in one go for the duration of the Stata session. If you want to permanently force Stata to do this for all future sessions, type:

```
set more off, permanently
```

Alternatively, if you want Stata to always wait for you to click through the results, type:

```
set more on [, permanently]
```

## 2.2 Summarize

**summarize** (note the American spelling, short: **sum**) gives a summary of one or more variables:

**summarize**

Variable	Obs	Mean	Std. Dev.	Min	Max
var1	1,000	500.5	288.8194	1	1000
age	1,000	126.546	257.962	20	999
sex	0				
height	957	168.6917	8.556731	149	192
weight	944	169.7401	26.3858	95.90097	246.9174

This will display a summary of all variables in your dataset, telling you how many non-missing (numeric) observations there are for each variable, its mean value, standard deviation, and minimum and maximum values. String variables, like *sex*, will have 0 observations and no other information (you can't find the mean of a string, for example).

Adding the option **detail** gives you more detailed information, including the median and other percentiles, while adding variable names after **summarize** will restrict the summaries to those variables in the same way it did for **list**. For instance, if we wanted to know the median of age (the 50<sup>th</sup> percentile), we would write:

**sum age, detail**

Age (years)				
Percentiles		Smallest		
1%	20	20		
5%	23	20		
10%	27	20	Obs	1,000
25%	36.5	20	Sum of Wgt.	1,000
50%	53		Mean	126.546
		Largest	Std. Dev.	257.962
75%	70	999	Variance	66544.41
90%	79	999	Skewness	3.072292
95%	999	999	Kurtosis	10.49753
99%	999	999		

We can see that some ages are 999 (the largest four values at least, and the top 5%), which seems implausible when age is measured in years. We'll change this in just a bit to a missing value, as 999 is a common code for missing values.

Like most commands, we can shorten the **summarize** command to **sum**, and we often use it to have a quick look at the data before manipulation.

## 2.3 Tabulate

The command **tabulate** (short: **tab**), can produce one- and two-way tables of categorical variables (variables categorized into groups, like hair colour or sex), depending on whether one or two variables are specified when you run the command. If we wanted to see the number of observations for each hair colour in a one-way table, we would type:

```
tabulate hair_colour
```

hair_colour	Freq.	Percent	Cum.
Black	199	20.29	20.29
Blonde	288	29.36	49.64
Brown	294	29.97	79.61
Grey	97	9.89	89.50
Red	103	10.50	100.00
Total	981	100.00	

In addition to the number of observations, we see percentages and the cumulative percentage. By default, missing values are left out of tabulations, so you can see only 981 out of 1,000 people have values of hair colour.

If we wanted to see the number of observations for each hair colour, but this time split by smoking status in a two-way table, we would type:

```
tab hair_colour current_smoker
```

hair_colour	Smoking status, 1 = Yes, 2 = No		Total
	1	2	
Black	67	130	197
Blonde	91	194	285
Brown	97	196	293
Grey	36	60	96
Red	37	65	102
Total	328	645	973

Here, we see only the number of observations within each hair colour for each smoking status. For example, there are 67 people who have black hair and smoke (smoke == 1), and 130 people who have black hair who do not smoke (smoke == 2), adding up to 197 people with black hair. The total number of observations has dropped to 973, as anyone with a missing hair colour *or* smoking status will be omitted from the table.

If we want to see the missing values, we can add the option **missing** to the command:

```
tab hair_colour current_smoker, missing
```

hair_colour	Smoking status, 1 = Yes, 2 = No			Total
	1	2	.	
	6	13	0	19
Black	67	130	2	199
Blonde	91	194	3	288
Brown	97	196	1	294
Grey	36	60	1	97
Red	37	65	1	103
Total	334	658	8	1,000

The missing hair colours are blanks (top row), and the missing smoking statuses are represented by a full stop (the missing value character for numeric variables in Stata, penultimate column).

Two-way tables don't show percentages by default: it wouldn't necessarily be clear whether the percentages were going across the rows, e.g. percentage of people with black hair who smoke, or down the columns, e.g. percentage of people who smoke who have black hair. However, we can add the percentages with the **row** and **column** (short: **col**) options, saying that we want to see the row and column percentages respectively:

```
tab hair_colour current_smoker, row
```

hair_colour	Smoking status, 1 = Yes, 2 = No		Total
	1	2	
Black	67 34.01	130 65.99	197 100.00
Blonde	91 31.93	194 68.07	285 100.00
Brown	97 33.11	196 66.89	293 100.00
Grey	36 37.50	60 62.50	96 100.00
Red	37 36.27	65 63.73	102 100.00
Total	328 33.71	645 66.29	973 100.00

```
tab hair_colour current_smoker, column
```

hair_colour	Smoking status, 1 = Yes, 2 = No		Total
	1	2	
Black	67 20.43	130 20.16	197 20.25
Blonde	91 27.74	194 30.08	285 29.29
Brown	97 29.57	196 30.39	293 30.11
Grey	36 10.98	60 9.30	96 9.87
Red	37 11.28	65 10.08	102 10.48
Total	328 100.00	645 100.00	973 100.00

You can specify both row and column if you like, but it does get a little confusing.

When you tabulate labelled numeric variables, like *accommodation*, then you see their labels rather than their underlying numbers:

**tab accommodation**

Accommodation: 1 = Own, 2 = Mortgage, 3 = Rent	Freq.	Percent	Cum.
Own	294	31.24	31.24
Mortgage	184	19.55	50.80
Rent	463	49.20	100.00
Total	941	100.00	

However, you can force Stata to show you the underlying numbers by adding the **nolabel** (short: **nol**) option:

**tab accommodation, nolabel**

Accommodati			
on: 1 =			
Own, 2 =			
Mortgage, 3			
= Rent	Freq.	Percent	Cum.
1	294	31.24	31.24
2	184	19.55	50.80
3	463	49.20	100.00
Total	941	100.00	

This can be useful if you need to quickly check which label is assigned to which number, but it can also be useful to see them both at the same time: as this involves modifying **value labels**, which we do in **Section 2.8 Labelling Values**.

## 2.4 Generating Variables

You can create new variables in Stata using the **generate** command (short: **gen**), which has the syntax:

```
generate {new variable name} = {some expression}
```

Where **{some expression}** is an expression that Stata understands, which can be as simple as adding two variables together or multiplying a variable by 2, or could be as complex as you can imagine: my largest **generate** command stretched to 10 lines when I copied it into a word document. Remember that the **{new variable name}** must start with a letter, have no spaces and be unique, i.e. not named after any existing variables.

Let's suppose that we want to create a new variable that is the mean of systolic and diastolic blood pressure (I'm pretty sure this is clinically meaningless, but let's suppose anyway):

```
generate average_bp = (systolic_bp+diastolic_bp)/2  
(73 missing values generated)
```

The only output from this command is to tell us if any observations in the new variable are missing. In this example, the new variable *average\_bp* will be missing for any observations that are missing for either systolic or diastolic blood pressure, since missing values aren't used in most calculations in Stata. This is the usual cause of missing values, though Stata will also give missing values for calculations that don't make sense, like dividing a number by 0. Brackets follow BODMAS rules, i.e. brackets first, then orders (squares, cubes etc.), division, multiplication, addition and finally subtraction. As ever, it's usually a good idea to take a look at new variables to make sure they're doing what you want.

When writing out expressions, spaces don't matter all that much: they aren't necessary, but they also won't mess up a command if they're there. If things look clearer to you with spaces, put them in. If they don't, then don't.

Let's add another variable for practice. This time, let's generate another ID number equal to the observation number (there's already an ID variable that does this, *var1*, but let's pretend we didn't see that):

```
gen id2 = _n
```

The **\_n** expression tells Stata to use the row number in each observation, so here the numbers 1 to 1,000 inclusive, and is a useful thing to know. There's no output for this command because there are no missing values in the expression.



## 2.5 Replacing Variables

Once a variable is created, we can't use **generate** to overwrite the data, as Stata requires a new variable name for each **generate** command. This is part of Stata's attempt to protect you from overwriting data: if you want to open a new dataset on top of a dataset already in memory, you either need to save your existing dataset or provide the option **clear**. Equally, if you want to save a dataset over an existing dataset, you need to provide the option **replace**.

If you want to overwrite a variable in a dataset, instead of using **generate**, you need to use the command **replace** (no short form, which may also be part of Stata's efforts to stop you accidentally deleting or overwriting data). The syntax, however, is the same as for **generate**:

```
replace {existing variable name} = {some expression}
```

Let's say we've noticed that height is in centimetres and we would really like it in metres:

```
replace height = height/100  
(957 real changes made)
```

Stata will tell you how many changes it has made when replacing a variable, in this case 957 changes (the 43 missing observations for height aren't changed). We've also noticed that weight is in pounds (lbs), and would like it to be in kilograms (kg), and there are 2.20462 pounds per kilogram:

```
replace weight = weight/2.20462  
(944 real changes made)
```

Now we have height in metres and weight in kilograms, we could also generate a new variable for body mass index (BMI), which is weight in kg divided by height in metres squared (the <sup>^</sup> symbol is used for powers/orders, so  $x^2$  means  $x$  squared,  $x^3$  means  $x$  cubed etc., where  $x$  can be a variable or a number):

```
gen bmi = weight/height^2  
(95 missing values generated)
```

Let's do a quick check of BMI to see whether the command has generated something that looks reasonably correct:

```
sum bmi
```

Variable	Obs	Mean	Std. Dev.	Min	Max
bmi	905	26.98933	3.204688	17.42509	37.25286

The summary of BMI tells us the average BMI is about 27 kg/m<sup>2</sup>, which seems about right, and the minimum and maximum values are also about right for BMI. There are 95 missing values, but that's ok, there were quite a few missing values for height and weight too.

## 2.6 Renaming Variables

Renaming variables requires another new command, this time **rename**, which has the syntax:

```
rename {old variable name} {new variable name}
```

The limitations on naming variables still apply: it can't be the same as an existing variable, can't have special characters (apart from an underscore), and you have to start a variable name with a letter (or underscore).

In our dataset, the original ID number is named *var1*, so we will rename it *id*, and whether someone has run a marathon in the past year is named *var13*, so we'll rename that *marathon*:

```
rename var1 id  
rename var13 marathon
```

## 2.7 Labelling Variables

Labelling variables is usually a good idea as it gives more context and information to variable names. In the top right variables window, you can see the variable labels next to the variable names. Currently, *hair\_colour* and each of the variables we've just created are unlabelled, so if we come back to the dataset in a year, we may forget what each variable actually is.

Labelling variables is straightforward, but note the quotation marks around the label:

```
label variable {variable name} "{Label of your choice}"
```

Let's add variable labels to every variable that doesn't have a label:

```
label variable hair_colour "Hair colour"  
label variable average_bp "Average of systolic and diastolic  
blood pressure (mmHg)"  
label variable id2 "ID number (2)"  
label variable bmi "Body Mass Index (kg/m2)"
```

Arguably, *hair\_colour* and *id2* don't need labels, but who knows what we'll think these variables mean in a year. You can make labels up to 80 characters long, and because they're strings, there isn't a limit on the characters you can use. If you don't specify a label it will remove the variable label. For example, typing `label variable bmi` would remove the label for *bmi*.

Also, we've noticed that because we changed the units of height and weight, their labels are now wrong. Let's fix that:

```
label variable height "Height (m)"  
label variable weight "Weight (kg)"
```

This will replace the height and weight labels: we're overwriting metadata (data about our data), not the data itself, and I guess Stata is less concerned about that than overwriting data.

## 2.8 Labelling Values

Labelled numeric variables receive their labels through the same **label** command, but it's used in a different way. Remember that the labelled numeric variables appear **blue** in the spreadsheet view, have labels that look like strings but act as numbers when using them in commands and expressions. These are particularly useful for categorical and binary variables, where the numbers mean something, for example 0 = "No" and 1 = "Yes".

This is a two-stage command, and so slightly trickier than previous commands. First, we need to define a label and give it both a name and a list, telling it what each number (which has to be an integer, i.e. whole number) in the label should mean. Labels are separate from variables, so you normally only see them through their effects on variables. The label names should be unique (they can be the same as variable names, just unique for label names) and start with a character, and you can have up to 1,000 (or 65,536 depending on your version of Stata) numbers with associated meanings per label. Once defined, we then apply the label to one or more variables.

The syntax is as follows:

```
label define {label name} {1st number} {"meaning of 1st number"}  
{2nd number} {"meaning of 2nd number"} ... {last number} {"meaning  
of last number"}
```

```
label values {list of variable names} {label name}
```

Let's do this for both *marathon*, where 0 = "No", 1 = "Yes", and *current\_smoker*, where 1 = "Yes" and 2 = "No":

```
label define marathon_label 0 "No" 1 "Yes"  
label values marathon marathon_label
```

```
label define smoker_label 1 "Yes" 2 "No"  
label values current_smoker smoker_label
```

There's no output from these commands, but you can see their effects when you switch to the spreadsheet view:

marathon	current_sm~r
.	No
No	No
No	Yes

The *marathon* and *current\_smoker* variables are now **blue**, so are now labelled numeric variables rather than simply numeric variables. We can also see this if we **tabulate** the two variables:

```
tab marathon current_smoker
```

Ran a marathon in past year	Smoking status, 1 = Yes, 2 = No		Total
	Yes	No	
No	262	493	755
Yes	64	142	206
Total	326	635	961

If you want to be able to see the underlying number *and* the label too, you can use the **numlabel** command, which adds or removes the underlying number as a prefix to the labels:

```
numlabel marathon_label smoker_label, add
tab marathon current_smoker
```

Ran a marathon in past year	Smoking status, 1 = Yes, 2 = No		Total
	1. Yes	2. No	
0. No	262	493	755
1. Yes	64	142	206
Total	326	635	961

This can make it easier to see what the underlying numbers are for any numeric labelled variables. However, if you tire of this and want to remove the prefixes, you can, again using **numlabel**:

```
numlabel marathon_label smoker_label, remove
tab marathon current_smoker
```

Ran a marathon in past year	Smoking status, 1 = Yes, 2 = No		Total
	Yes	No	
No	262	493	755
Yes	64	142	206
Total	326	635	961

If you want to list all the labels in memory, type:

```
label dir
```

```
smoker_label
accommodation
_merge
marathon_label
```

This shows you all the labels you have in memory. The labels are saved with the dataset, so you don't need to worry about losing them. If you want to list all the labels in memory along with their contents, type:

```
label list
```

```
smoker_label:
    1 Yes
    2 No
marathon_label:
    0 No
    1 Yes
accommodation:
    1 Own
    2 Mortgage
    3 Rent
_merge:
    1 master only (1)
    2 using only (2)
    3 matched (3)
    4 missing updated (4)
    5 nonmissing conflict (5)
```

If you want to remove any labels, you can type:

```
label drop {label name}
```

For example, the `_merge` label, which was created when we merged datasets earlier, can be removed:

```
label drop _merge
label list
```

```
smoker_label:
    1 Yes
    2 No
accommodation:
    1 Own
    2 Mortgage
    3 Rent
marathon_label:
    0 No
    1 Yes
```

Finally, you can add to or modify existing labels using **label define** and the **add** and **modify** options. Adding a new number-meaning pair is easy enough with the **add** option:

```
label define smoker_label 0 "Added label", add
label list
```

```

smoker_label:
    0 Added label
    1 Yes
    2 No

```

You can see this has added the **0 Added label** pair to the *smoker\_label*.

You can't add a number-meaning pair that already exists – this is like generating a variable that already exists, Stata won't let you do it because you're overwriting data, instead of generating new data. Rather, if you want to change a number-meaning pair, you can with the **modify** option:

```

label define smoker_label 0 "Modified label", modify
label list

```

```

smoker_label:
    0 Modified label
    1 Yes
    2 No

```

Finally, if you want to remove a particular number-meaning pair, you can modify it but leave the meaning blank:

```

label define smoker_label 0 "", modify
label list

```

```

smoker_label:
    1 Yes
    2 No

```

## 2.9 Removing Data

Removing data in Stata is straightforward: use the **drop** command to remove any unwanted variable(s):

```
drop {variable(s) to be removed}
```

Dropping a variable is permanent, so it is important to be certain you are removing the correct variable. This is also why it makes sense to have a master copy of a dataset that you don't save over and use do files to make all the changes to a dataset, so you can always reopen the master dataset and redo all the commands if you make a mistake.

In our dataset, let's get rid of the *\_merge* and *id2* variables, since we don't need them anymore:

```
drop _merge id2
```

Assuming the variables you've specified exist, Stata doesn't give you any notification that the variables have been removed, but if you check either the variables window in the top right, or the spreadsheet view, you'll find the variables are gone.

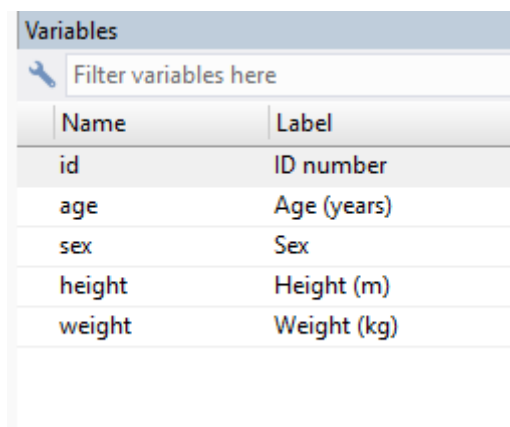
Now save your dataset with your initials and the lesson number, because we are about to start the exercises for this lesson, and we'll need this dataset in the next lesson:

```
save "BP_exercise_SH_02.dta", replace
```

Just before we go onto the exercises though, let's introduce the **keep** command: **keep** is the opposite of **drop**, in that **keep** keeps everything you specify and drops all other variables, and **drop** drops everything you specify and keeps all other variables. The syntax is the same.

For instance, if we decided we only needed the variables *id*, *age*, *sex*, *height* and *weight*, then we could use the command:

```
keep id-weight
```



Variables	
Filter variables here	
Name	Label
id	ID number
age	Age (years)
sex	Sex
height	Height (m)
weight	Weight (kg)



Rather than writing out all the variables you want, you can use a quick trick: when listing variables, you can say you want two variables and all variables in between them by putting a dash between them. This trick is dependent on the variables being ordered correctly, so it's best to be sure you are specifying all the variables you want when doing this.

If you want to remove all variables and all observations, you can use another trick:

**drop \_all**

In many commands, Stata understands **\_all** to mean all variables. Dropping all observations and variables, however, doesn't drop labels, which persist. To get Stata back to a clean slate, as if you'd closed and reopened it, use:

**clear**

This will reset Stata entirely, ready for new data. This is the same as the option you specify when loading in a new dataset on top of existing, unsaved data. Stata is forcing you to explicitly say **clear** so it can be confident you're not overwriting data you actually need.

Finally, **drop** can be used to remove observations as well as variables, but this requires using *if* or *in* statements, which we cover in the next lesson.

## 2.10 Exercise

For this exercise, see how you do with the following:

1. Load in the **Exercise\_01\_{initials}.dta** dataset from the first exercise
  - a. If you're not confident with your dataset, you can load in the **Exercise\_02.dta** dataset instead
2. List the values of *age* and *height* together
3. Summarise the dataset, then summarise *height* in detail
  - a. Check to see if anything looks wrong here
4. Tabulate *hair\_colour* and find out how many people have red hair
5. Tabulate *hair\_colour* and *accommodation* and find out what percentage of people with black hair rent
6. Tabulate *hair\_colour* and *accommodation* and find out what percentage of people who own their home have grey hair
7. Add number prefixes to the *hair\_colour* and *accommodation* labels, and tabulate *hair\_colour* and *accommodation* again to check it worked
  - a. You'll need to list the labels to find out to which labels you'll need to add the number prefixes
8. Modify the *hair\_colour* label so "Blonde" is spelled correctly
  - a. Check the label to make sure the label is fixed
  - b. See if you can spot a new problem with it: if so, see if you can fix it
9. Convert *age* to years and *weight* to kilograms (1 stone = 6.35 kg)
10. Generate body mass index (BMI: weight in kilograms divided by height in metres squared), then give it an informative variable label
  - a. Summarise the variable to check it looks about right
  - b. If not, think about why they might be wrong
11. Create a label for *income* and apply it to the variable using these number-meaning pairs, then check it worked:
  - a. 1 = <£18,000
  - b. 2 = £18,000 to £30,000
  - c. 3 = £30,000 to £50,000
  - d. 4 = £50,000+

12. Rename *var3* and *var14* to something more informative using the variable labels as a guide
13. Remove the *\_merge* variable (if present)
14. Save the modified dataset, calling it **Exercise\_02\_{initials}.dta**

## 2.11 Exercise – Answers

1. We've done this a few times now:
  - a. `use "Exercise_01_SH.dta", clear`
  - b. Or, if you're not confident in your save from exercise 1:  
`use "Exercise_02.dta", clear`
2. `list` command:
  - a. `list age height`
3. `summarize` command:
  - a. `sum height, detail`
  - b. There are values of -1 for height, which definitely seems wrong – these are likely missing value codes again
4. A series of `tabulate` commands now:
  - a. `tab hair_colour`
  - b. 141 people have red hair
5. Add the `row` option to see the percentages across rows (if you specified *hair\_colour* and *accommodation* the other way round, you'll need the `column` option)
  - a. `tab hair_colour accommodation, row`
  - b. 48.36% of people with black hair rent
6. Add the `column` option to see the percentages down columns (if you specified *hair\_colour* and *accommodation* the other way round, you'll need the `row` option)
  - a. `tab hair_colour accommodation, col`
  - b. 12.06% of people who own their home have grey hair
7. We'll need to check the label names, then use `numlabel` for this:
  - a. `label list`
    - i. The *hair\_colour* and *accommodation* labels are called, simply, *hair\_colour* and *accommodation*
  - b. `numlabel hair_colour accommodation, add`
  - c. `tab hair_colour accommodation`

8. This seems simple, but the number prefixes make it complicated:
  - a. `label define hair_colour 3 "Blonde", modify`
  - b. `label list`
    - i. The "Blonde" label doesn't have a number prefix, while the others do.
  - c. We can get around this in a couple of ways (either works fine):
  - d. `numlabel hair_colour, add force`
    - i. The **force** option gives number prefixes to those without them
  - e. `label define hair_colour 3 "3. Blonde", modify`
    - i. This also works, it just adds the prefix directly
  
9. **replace** commands:
  - a. `replace age = age/12`
  - b. `replace weight = weight*6.35`
  
10. You'll need a few commands here:
  - a. `gen bmi = weight/height^2`
  - b. `label variable bmi "Body mass index (kg/m2)"`
  - c. `sum bmi, detail`
    - i. Some values are very high (>100): this is likely due to the "-1" values of *height*, and we'll fix this in the next lesson
  
11. Create and apply a new label:
  - a. `label define income 1 "<£18,000" 2 "£18,000 to £30,000" 3 "£30,000 to £50,000" 4 "£50,000+"`
  - b. `label values income income`
  - c. `tab income`
  
12. **rename** commands:
  - a. `rename var3 sex`
  - b. `rename var14 smoking_status`
    - i. Or *current\_smoker*, or anything really so long as it makes sense
  
13. **drop** command:
  - a. `drop _merge`
  
14. And finally, a **save** command:
  - a. `save "Exercise_02_SH.dta", replace`