** IPA/J-PAL STAFF TRAINING**

**STATA 101**

TABLE OF CONTENTS

[INTRODUCTION: 2](#_Toc379316468)

[Chapter 1. THE BASICS: WHAT STATA LOOKS LIKE 3](#_Toc379316469)

[OPENING A DATASET: 3](#_Toc379316470)

[COMPARING WITH EXCEL: 4](#_Toc379316471)

[Chapter 2. LOOKING AT YOUR DATA: SOME BASIC COMMANDS 5](#_Toc379316472)

[STRUCTURE: 5](#_Toc379316473)

[SHORTHAND: 5](#_Toc379316474)

[1. SUMMARIZE 5](#_Toc379316475)

[2. TABULATE 6](#_Toc379316476)

[3. LIST 7](#_Toc379316477)

[Chapter 3. IMPOSING CONDITIONS 7](#_Toc379316478)

[1. IF 7](#_Toc379316479)

[2. AND/OR SYNTAX 8](#_Toc379316480)

[Chapter 4. SAVING AND SORTING 11](#_Toc379316481)

[1. SAVE 11](#_Toc379316482)

[2. SORT 11](#_Toc379316483)

[Chapter 5. MANIPULATING YOUR DATA: CREATING AND CHANGING VARIABLES 13](#_Toc379316484)

[1. GENERATE 13](#_Toc379316485)

[2. REPLACE 13](#_Toc379316486)

[3. DROP 15](#_Toc379316487)

[4. SAVE, revisited 15](#_Toc379316488)

[EX 1. EXERCISES FOR CHAPTERS 1-5. 15](#_Toc379316489)

[Ex 2. ADDITIONAL REVIEW 17](#_Toc379316490)

[Chapter 6: DO-FILES 18](#_Toc379316491)

[1. WHAT IS A DO-FILE AND WHY DO WE NEED ONE? 18](#_Toc379316492)

[2. STARTING A DO-FILE 19](#_Toc379316493)

[3. EXECUTING COMMANDS 19](#_Toc379316494)

[Chapter 7: REPRODUCIBILITY: A QUICK GUIDE 20](#_Toc379316495)

[1. FOLDER STRUCTURE AND ORGANIZATION 20](#_Toc379316496)

[2. MASTER DO-FILE 21](#_Toc379316497)

[3. ANNOTATION 21](#_Toc379316498)

[4. TYPICAL OPENING COMMANDS 22](#_Toc379316499)

[*A.* CLEAR 23](#_Toc379316500)

[*B.* SET MORE OFF 23](#_Toc379316501)

[*C.* SET MEMORY 23](#_Toc379316502)

[4. OTHER GOOD COMMANDS TO INCLUDE IN THIS INITIAL SECTION: 23](#_Toc379316503)

[*A.* VERSION OF STATA 23](#_Toc379316504)

[*B.* LOG CLOSE 23](#_Toc379316505)

[SUMMARY 24](#_Toc379316506)

# INTRODUCTION:

Welcome to IPA/J-PAL’s introduction to Stata, Stata 101. This course is meant for those with no previous experience with Stata—we start from the beginning. As you progress, you will learn the various ways that IPA projects can use Stata to handle and manage their data.

All the answers to the exercises are in the soft copy of this document, in white-colored font. So, wherever you see a place where it says “Answer” or “A:” followed by a blank space, you can highlight it and change the font color to black to see the answer. Please only do this to check your answers. This module is meant to be interactive. It is strongly recommended that you actually try every step of the directions in Stata, not simply say, “that sounds straightforward,” and skip over it.

Continue learning Stata by visiting our [Stata training resource folder](https://ipastorage.box.com/stata-resources) on Box.

# Chapter 1. THE BASICS: WHAT STATA LOOKS LIKE

## OPENING A DATASET:

1. Now we will open the dataset we will work with. Find the file named “***intro.dta***”, located on your USB under (on your USB) in Stata > 101 > Data. Double click on it. Stata will automatically launch.
   * If Stata is already running, a new Stata window will open.
   * Alternatively, if Stata is running you can go to *File* → *Open**→ Choose your dataset*
2. Notice that there are four sections, or “windows”: Review, Results, Command, and Variables.

These sections might look slightly different in Stata 11, 12, and 13, but all will be clearly labeled.



**REVIEW 🡪**

**🡨 RESULTS**

**VARIABLES 🡪**

**🡨 COMMAND**

* **command:** You can tell Stata what to do by typing in commands. Click inside the command window and type

display “Hello”

* **results:** Here Stata displays the commands followed by the output that Stata has produced. *Note what appeared as the result of the command you just typed in.*
* **variableS:** lists all the variables in the dataset. The variable window can act as a shortcut for creating commands. Try clicking on one of the variables. It should appear in the command window, eliminating the need for you to write it out.
* **review:** Lists all your prior commands. Notice that **display “**Hello!**”** now appears there. You can click on it and it will appear in your command window.

***Useful tip:*** *When you are in the command window, you can scroll through your previous commands using the* ***Page Up*** *and* ***Page Down*** *buttons on your keyboard.*

As you can guess from a glance at the variable list, this dataset contains math and reading scores, as well as attendance rates, of students in various schools. Specifically, the dataset covers one class per school in four different schools, 2 public and 2 private. In addition to scores, attendance and the school variable, it also contains information on name, ID and gender of the student.

## COMPARING WITH EXCEL:

* + - 1. Find the file named “intro.xls” on your USB (Stata > 101 > Raw). Double click on it. The file will automatically launch in Excel.

1. Compare variable names (school, student, name, math, reading, female, class) in Excel and Stata. See where they are located in Excel. Now compare with Stata.
2. Now let’s look at what a dataset stored in Stata actually looks like (the way Stata sees it and reads it). Type

**browse**

in the command window.

Notice that a window opens that resembles an Excel spreadsheet, although that you **cannot** actually work with the dataset or change it in that format. The columns show the variables and the rows show the observations.

Notice that “female” takes the value of either 1 or 0 (can you guess what it means for female to equal 0?). The variable “school” takes the value of 1 to 4, and “names” takes the form of text. Exit out of the browse window by clicking on the ‘X’ on the upper right hand corner. Notice that “browse” has appeared in your review window.

# Chapter 2. LOOKING AT YOUR DATA: SOME BASIC COMMANDS

*Now that you can open Stata and hopefully feel comfortable with its layout, it’s time to learn some very easy commands to get Stata to perform some basic analysis for you.*

STRUCTURE:The most basic commands that we’re about to show you follow a simple structure of:

**command**

*-or-*

**command** *variable*

where command is the action you want done and variable is the 1 or more variables you want it done to.

For example, you’ve already seen the command **browse**.

You can also browse specific variables. Try browsing the two variables:

**browse** *female school*

Here, you see browse is the command, and female and school are the variables of interest. In the browse window, only those two columns are shown.

***Note:*** *This is very useful when the dataset is large and you are trying to check something specific.*

SHORTHAND: Commands and variables can both be shortened. For most common commands the first 2 or 3 letters work. For example:

**br** *math*

Now, let’s become familiar with some of the most used and useful commands for looking at your data and producing some basic statistics about it.

1. SUMMARIZE **(**shortened: ***sum, summ*)** gives you basic statistics about variable(s). To look at summary statistics for ALL variables, simply type in:

**summarize**

As you can see, the output is the number of observations, mean, standard deviation, and the min. and max. of all variables in the dataset.

Now let’s try producing the summary statistics for just one variable. This is an extremely useful function, both to use during cleaning and for production of summary statistics (for instance, for the baseline). Let’s get the summary stats for the math score. Type:

**summarize** *math*

****Something like this table should pop up:

For how many students do we have scores? What is the mean of math scores across the students? What is the standard deviation? What was the lowest score? Please check your answer after you come up with it.

*A: 39 scores, 81.6 mean, 18.9 std. dev., and 38 as the lowest score.*

Now produce the same statistics for the variable of reading scores.

*A: 36 scores, 80.1 mean, 22.5 std. dev., and 27 as the lowest score.*

***Note:*** *Sometimes you also need more detail on the variable than just the number of observations-mean-sd-min-max. That is why the summarize command has a “detail” specification. Try this:*

**summarize** *math*, detail

*What happened? As you can see, it also produces a percentile breakdown of observations and other stats, like variance, that can sometimes come in handy.*

1. TABULATE(shortened: **tab**) -lists all the values the variable takes in increasing (or alphabetical) order, tells you how many of each value there are and what percentage each value constitutes. Type in:

**tab** *school*

As you can see, tab produced a list of number and percentage of students by school. How many students were observed from School 1? What percentage of total observations was from School 3?

*A: 9 students in School 1, 25.64% of the total in School 3.*

***IMPORTANT!***  Sometimes a value of a variable will be missing (i.e. there’s no data). This is quite common. For instance, scroll back to where you summarized math and reading in the results window in Stata. Compare the total number of observations for reading scores with that for the math scores. What is the difference? Please continue to check your answers after the exercises.

*A: There are 3 more observations for math than reading.*

Tabulate (and many other commands) doesn’t automatically show you the missing values, so it is easy to forget about their existence. However, doing so sometimes gives you inaccurate stats. To see the missing values when you tabulate type:

**tab** *reading,* missing

*-Compare this with -*

**tab** *reading*

As you might have figured out, the missing values for numeric variables get coded as “.” (a dot). Notice that there are three missing values in “reading.” You can see them at the very bottom of the list. Stata sees missing values as the biggest value possible and always will list them last.

You can also check this by typing:

**browse** *reading math*

and confirming that there are missing variables. However, this is not useful if your dataset is large.

**Always**, always **keep in mind the missing values** when doing any coding. Always ask – how will this piece of code affect missing values? For example, if you ask Stata to find the mean of math scores for students with attendance rates above 80%, those with missing values for attendance will be included in this analysis. Would you have wanted this?

1. LIST **–** lists the values of variables (in the order they are in the dataset, repeating any duplicates)

**list**

**list** *reading*

# Chapter 3. IMPOSING CONDITIONS

*Knowing what the average math or reading score is undoubtedly useful, but it is not the end of the story. More often than not, it is not the total mean of the dataset that matters, but that of a subset of the observations. The most typical in our trade is that of looking at the treatment v. control group means. Others that might be useful for this dataset (a baseline) is a breakdown by gender, class, or both. Stata offers multiple ways to impose conditions on the command. Here are the two simplest and most commonly used:*

* + - 1. IF– a condition that qualifies any command to which it is applied. The command is only performed on observations for which the condition is met.

For example, imagine we wanted the average of the girls’ math score – i.e., the average of the scores of **only** girls.

For this we would want to look at individuals who meet the condition of being female. In other words, we would want: **female == 1**. So, try:

**summ** *math* **if** *female* == 1

***Note*** *the* ***double equal (==) sign****. This is the way Stata signals a condition is imposed, meaning that you want to look at a variable that already is equal to 1 (or whatever else), and that you’re not changing the variable. Later, you will see how to generate variables. To do this and set a variable equal to something, you will use one equal (=) sign.*

*Always use a double equal sign when summarizing, tabulating, or imposing conditions. If you just put one equal sign instead of two Stata will give you an error. You can try it to see what Stata error feedback looks like.*

What did you find to be the average math score for girls?

*A: 80.1*

Now find what the average score is for boys. First, what is the Stata condition of being male?

*Hint: remember, there is no* ***male*** *variable, so you have to use the female one.*

*A: female* ***== 0***

Now how would you find the average score for the boys using that condition?

*A:* **summ** *math* **if** *female* == 0*; the answer is 83.3*

How do boys compare with girls?

*A: The boys’ score is higher, although it does not seem significant judging by the standard deviations.*

For another example, let’s see how this works with **COUNT,** a command that counts the number of observations that satisfy the specified conditions. (If no conditions are specified, count displays the number of observations in the data. Type in **count** to see what it does. You should get the total number of observations in the dataset. *A: 39*).

Now try:

**count if** *female* == 0 to see the number of boys in the dataset.

*A: 18*

***Note: the “in” qualifier***

Another useful qualifier is the **in** qualifier. “In” at the end of a command means that the command is to use only on the (range of) observations specified. For example, try:

**list** *reading* **in** 1/5

As you can see, the command listed the reading scores for only the first five observations in the dataset, instead of all the observations. When could this be useful?

1. AND/OR SYNTAX– used to apply multiple conditions at the same time

Suppose you wanted to see the math scores not just for girls, but only for girls in school 3. Or you wanted to know what the math score average was in schools 1 and 2 (but not 3 and 4). Or supposed you wanted to get really fancy and wanted to see the math score averages for girls in schools 1 and 2. How would you go about that? This is where and/or syntax comes in.

Suppose you have two Conditions: A and B.

**AND** is used when both Condition A and Condition B have to be met. In the illustration below, imposing the A “AND” B condition gives us the purple-colored intersection between the two circles.

AND is coded as the symbol “**&”** in Stata. Stata doesn’t recognize “and” (spelled out) as a command and will not execute it.

So, if you wanted to meet both the Conditions A and B, you would input:

**command if** A == true **&** B == true

Using our previous example, to see the score for girls in school 3, think of being a girl as condition A and being in school 3 as condition B. We need the students to meet both conditions. To get people who fall in both groups you have to use the AND function:

**summarize** *math* **if** *female* **==** 1 **&** *school* **==** 3

What is the average math score for girls in school 3?

*A: 70.5*

**OR** is used by Stata to signify the group for which either of the conditions **(including both the conditions at the same time)** are met. In the illustration below, saying the person can meet the condition A or B would mean anything within the A and B circles (colored a pleasant violet color this time) is acceptable – that is anyone who is A, anyone who is B, and anyone who is both qualifies. However, note that things in C that are not also part of B are outside our condition.



Or is denoted by symbol “|” (made by shift + \).

So, to see the average of the math scores for schools 1 and 2, we say:

**summarize** *math* **if** *school* == 1 | *school* == 2

*What is the average?*

*A: 81.9*

Now that we have the basics down, you can combine these operators in various ways to create any combination imaginable.

***Note!*** Just as in math, certain operations overwrite others, so always use parentheses to make sure Stata does exactly what you want it to do. Until you feel like you know exactly how Stata reads each command, always check your output and make sure what happened is what you wanted to happen.

To illustrate a potential pitfall, imagine if we wanted to see the reading scores for people who have math scores less than 65 or more than 90, due to some interest in looking at outliers. Think about how you would do this.

*A: You would look at people with scores less than 65 OR more than 90. What if you said AND instead? The command wouldn’t work, because there are 0 people who have a score below 60 and above 90 at the same time – that’s impossible!*

**summ** *reading* **if** *math* < 65 | *math* > 90

Now, what if you wanted to see this only for school 1? Use parentheses to delineate very clearly for Stata what you want it to do. Basically, you want it to look at the people who have the necessary scores AND who are in school 1.

To do this you would separate the condition of ‘scores’ from the ‘school 1’ condition using parentheses

**summ** *reading* **if (***math* < 65 | *math* > 90) & school == 1

If you didn’t put the parentheses in their proper place, Stata would read it as you wanting to summarize the reading score for all people who have a math score below 65 (regardless of school) and the separate group of those who are in school 1 AND have a math score over 90.

To try another one, let’s look at getting the average math score for girls in schools 1 and 2.

The answer is below, but try thinking it through yourself first. Sketch out a Venn diagram if you need it to help you write the code.

*A:* **summ** *math* **if** *female* == 1 **&** **(***school* == 1 **|** *school* == 2**)**

What would be the result if you forgot the parentheses?

*A: Stata will think you want to see all females in school 1 plus everyone in school 2, which makes no sense. So be careful not to do that!*

# Chapter 4. SAVING AND SORTING

*Now that you’ve learned how to look at your data and create some very basic statistics about it without changing it, it is time to learn how to “manipulate” your data—meaning change the data, create new variables, clean, and do everything else required. First, however, we need to learn how to save the dataset properly, so any changes you make can be preserved, but the mistakes you commit aren’t permanent.*

1. SAVE **–** saves the dataset

***IMPORTANT!***

Before you start messing with the dataset, you have to remember this: **DO NOT EVER OVERWRITE THE ORIGINAL DATA!**

This means that you do not ever make modifications to the original data and then save them in the same dataset.Imagine you made a mistake in coding and then saved the dataset with that mistake. What are you going to do then?

***Note:*** always keep extra untouched copies of the raw data in a separate folder just in case

To avoid any issues with overwriting, save the dataset under a different name (create a 2nd, modified version of the data). Please put it in the Data folder, together with the original dataset you are using. You should do this every time you do significant modifications to the data, and always do it if you work with the raw data. To do this say

**save** “intro\_modified.dta”**, replace**

***Note:*** If you just say **save** Stata will simply save over the dataset that’s open. The “replace” option after the comma specifies that if there is already a dataset with that name in the location, Stata will overwrite it with the current dataset instead of creating a new one.

If you are working with a dataset that is ok to modify (for instance, you saved it in the beginning of your work under a different name and now want to save all the changes you made), you can just say **save, replace** without further specifications.

1. SORT– arranges the observations of the current data into **ascending** order based on the values of the variables you list after the command. Try the following exercise:

**Browse** the data. Notice that right now the data is sorted by school (i.e. the observations are ordered by school number, lowest to highest). Suppose we wanted Stata to look at the data in the order of the student number instead. To accomplish this, say:

**sort student**

Now browse again. You can see the data is sorted by the student number. What if you wanted Stata to sort by student number within each school? You simply list the largest category first, and then add extra variables to the command to sort within the bigger categories. So, to sort by student number within schools, you’d type:

**sort** *school student*

Browse to see what that did. There is actually no limit to the number of variables that you can sort by at one time, and Stata just reads them left to right and sorts in that order.

Certain commands will require you to sort the data beforehand, and sometimes it’s a great trick to use with commands that don’t require it. However, you have to be very careful when using sort.

***IMPORTANT!***  If you sort by a non-unique set of variables (for example, by a household ID instead of an individual one), observations within that non-unique id group (in this case household ID) are sorted randomly. So if you sorted by household, within household one three individuals (1, 2, and 3) might get randomly sorted (listed) as 1,3,2 one time, and 3,1,2 another time.

What do we mean by that and how would that work for this dataset? Suppose your data was not sorted by student number (imagine it didn’t exist), and you sorted it by school. Although your school id would be sorted in order each time, the individuals within the school would be randomized. So if you had individuals 1,2,3 belong to school 1 and individuals 4,5 belong to school 2, if you just randomly kept sorting by school (by saying “**sort** *school”*), and then listed them, it might sort it as 3 different things 3 different times, even though the command remains the same:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1st sort | | 2nd sort | | 3rd sort | |
| school | individual | school | individual | school | individual |
| 1 | 1 | 1 | 3 | 1 | 2 |
| 1 | 2 | 1 | 1 | 1 | 1 |
| 1 | 3 | 1 | 2 | 1 | 3 |
| 2 | 4 | 2 | 4 | 2 | 5 |
| 2 | 5 | 2 | 5 | 2 | 4 |

So if you then created a command that, say, assigned new ids to those people based on the random order they were sorted into, their IDs could actually change every time you re-ran the program. This can end in a complete disaster resulting in months of extra work (true story). To avoid this make sure to sort by unique id or use “**sort** *school***, *stable*”**, which keeps the order random but the same every time you sort.

Please keep in mind that since your data is now sorted, this will not work right now. However, if you later open the original dataset and try this, you will see all the different patterns come up.

# Chapter 5. MANIPULATING YOUR DATA: CREATING AND CHANGING VARIABLES

*Now that you can save and sort, it is time to learn to create and change variables. This is a crucial step to master to become a real Stata user. You can’t get anywhere without variable generation!*

1. GENERATE– create or change contents of a variable.

**Generate (gen)** is an extremely easy command. You basically assign a value to the new variable that you’re creating, using conditions and pre-existing information in the dataset. Try this:

**gen** *test* **=** 1

You should see a new variable appear in your dataset, called “*test*”. If you **browse** you will see that this variable is uniformly equal to 1.

If Stata gives you an error saying “test is already defined”, this means that you’ve already created the variable (maybe you made an error the first time). See if the variable is correct. If it’s not and you have to re-do it, just name it something else other than “*test.*”

***Note*** that for generation, since it’s not a condition and you are in fact changing the variable, you only need one equal sign (=) instead of two, like for the if command.

Now let’s try imposing conditions for variable creating using the **if** function you just learned.

For instance, imagine that we just found out that two out of the four schools are private and two are not. So we want to generate a private school variable (let’s call it *private*). We know that school 3 is a private school, so for now let’s generate a variable for a private school that’s equal to 1 (meaning it’s true) for school 3.

**gen** *private*= 1 **if** *school* == 3

Now **browse** the two relevant variables *private* should be 1 when *school* is 3 and missing when it isn’t.

***Note:*** Stata recognizes capitalization, so ‘*Private’ is a different variable from ‘private.’*

1. REPLACE – modify contents of a variable

The same way generate creates a new variable equal to your specifications, replace does that for existing variables.

If you decided to change the existing variable, “*test*”, to 2 instead of 1, **generate** will not work. Stata will give you an error and tell you that this variable exists already. Instead you have to use **replace**. For instance, try:

**replace** *test* = 2

(and **browse**)

As you can see now, the values of the test variable that were equal to 1 before are now all equal to 2 (so you’re literally replacing the value of the variable with something else).

Now imagine that you learned that school 1 is private as well and you want to modify our private school variable(*private*)

**replace** *private* = 1 **if** *school* == 1

Finally, note that the*private*variable is blank when it’s not equal to 1. Now let’s set it to 0 for schools 2 and 4.

**replace** *private*= 0 **if** *school* == 2 **|** *school* == 4

Obviously, you are not restricted to just assigning uniform values. Stata would be a poor tool then. Rather, you can impose almost any condition imaginable, setting the variable equal to a modified version of itself or other variables (or a combination of thereof). What does this mean? It means that you can set a variable equal to itself times something, plus something, or multiplied by another variable, to give you an idea of just the simplest operations. You can also set a variable equal to another variable modified in whatever manner you choose.

For instance, suppose you realized that you had originally asked how frequently someone had done a task (like get water from a well) a day, and now you wanted a weekly measure. You could create a new variable with the old daily one multiplied by 7 (because there are 7 days a week) to create a variable that is a weekly measure.

To show you the mechanics, suppose we wanted to make the “*test*” variable twice as big as it had been before:

**replace** *test = test*\*2

**browse**

As you can see the *test* variable is now equal to 4. As is standard for these operations, \* stands for multiplication. You can also use / for division, ^ for raising to a power, etc.

***The “not” symbols***

Another important symbol to know is “not”, or the negative. For instance, suppose you want to make a variable equal to 1 when school is NOT equal to 3. To signify “not” Stata uses the symbols ~ (which is found to the left of your “1” key) and !. The two symbols are interchangeable for these purposes (although “!” is more commonly used because it is consistent with some other programming languages). To try this, let’s create a 2nd imaginary variable called “*test2*” that tags all schools except school 1:

**gen** *test2*= 1 **if** *school* **!=** 1

*or*

**gen** *test2*= 1 **if** *school* **~=** 1

1. DROP – eliminates variables or observations

Now suppose you want to delete a variable. In our case, we have the test and test2 variables that we definitely do not need and that clutter up the dataset. Use:

**drop** *test test2*

The command drop will delete the variable (or variables) you list; you can drop multiple variables at a time.

**KEEP** – **keep** is the opposite of **drop**, in that it keeps only the variables you list and deletes the rest of the variables in the dataset. It can be extremely useful when creating a smaller dataset with a subset of variables for faster work.

1. SAVE, revisited – now that we have the dataset saved as intro\_modified.dta and won’t be over-writing the original data, we can just save without specifications.

**save, replace**

***IMPORTANT!***  NEVER OVER-WRITE THE ORIGINAL DATA!

# EX 1. EXERCISES FOR CHAPTERS 1-5.

***All the answers are in white font below the questions, as before. There are also some hints you can highlight and see that are designated as “Hint”.***

**1.a.** Find the average math score for male students who are in private school. Call the new variable to do this *male\_pr* for “male in private school”

A: **gen** *male\_pr* = 1 **if** *female* == 0 & *private* == 1

**replace** *male\_pr* = 0 if *male\_pr* != 1

**sum** *math* **if** *male\_pr == 1*

*Average: 91.1*

*Hint:* First create a variable for male students in private school and then find the average.

To do this, generate a "tag" for male students who are in private school (a tag is a variable that tells you when something fulfills certain requirements; it’s usually 1 when the condition is true and 0 if not).

**1.b.** Now find out how the boys in private schools compare to the rest of the population (those who are not boys in private school), and to average score of other boys (not in private school).

A: **summ** *math* **if** *male\_pr*== 1

**summ** *math* **if fe***male*== 0 **&** *male\_pr* != 1

(for other boys only, average score of 77)

**summ** *math* **if** *male\_pr* !=1

(for everyone else, average score of 79.1)

**2.a.** Just like the math scores, the reading scores were supposed to have values from 0 to 100; however, if you summarize or tabulate the variable you’ll see that they instead range from 0 to 1. The percentages are the same but are out of 1 instead of out of 100. Try fixing it by adjusting the scale correctly:

Hint: Remember when we talked about how you can set the variable equal to another or to itself with modifications? (**replace** *test*= *test* \* 2)

A: **replace** *reading* = *reading*\*100

**2.b.** Now suppose we’re developing a program that wants to help students with failing reading scores (less than 65). Generate a tag for those failing students so we can identify our sample. Call it *failing\_reading*.

A: **gen** *failing\_reading* = 1 if *reading* < 65

**replace** *failing\_reading* = 0 **if** *failing\_reading* >= 65

***Note:***  There’s a more elegant and simple way to produce the same result in one line instead of two:

A: **gen** *failing\_reading2* = (*reading* < 65)

Stata interprets the line above as creating a tag. So when you say

**gen** *variable* = (condition), Stata will create a variable that is 1 when the condition is met and 0 when it’s not. In this case it will make *failing\_reading2* equal to 1 when the reading score is less than 65 and 0 when it’s not.

**2.c.** If you recall, reading scores were missing for some people that didn't show up for the test. What happened to the tag for those scores? What command would you use to check?

A: **tab** *failing\_reading* **if** *reading* == .

(**browse** also works)

**2.d.** As you can see, the missing scores automatically got coded as 0! In this case the missing values might be introducing bias, since it’s possible, for instance, that the students chose to skip the test because they knew they'd fail. In other circumstances you may want to code certain missings as something else. Just always make sure to account for them and to make sure you don’t include them in (or exclude them from) your code by accident.

As a recap, how did the missings get in there? Stata considers them the biggest value possible, so whenever you generate a variable basing it on whether the value of another variable is more than a set amount, you have to account for missing values.

In either case, we want to keep the tag as missing if the reading score was originally missing. Please fix the previous mistake.

A: **replace** *failing\_reading* = . **if** *reading* **== .**

**2.e.** In general, in order to avoid this problem, account for it whenever you’re generating variables. Try to generate failing\_reading3 as a new variable for failing reading using the most efficient method possible and accounting for the missing variables.

A: **gen** *failing\_reading3* = (reading<65) **if** *reading* **!= .**

**2.f.** Now imagine that instead of just helping the failing students, we wanted to split people into 3 achievement levels to best address each group's needs.

Generate a variable that divides the students into 3 tracked groups of different levels (call the variable *level*), those below 65, those above 90 and those in the middle. Assign the missings to the lowest level just for the purposes of this exercise.

A:  **tab** *reading*  → use this to eyeball groups of the same size

**gen level = 1 if reading < 65 | reading == .**

**replace level = 2 if reading >=65 & reading < 90**

**replace level = 3 if reading >= 90 & reading != .**

Hint: You don’t have to set the new generated variable equal to just 1 or 0. You can create as many groups as you want, just setting it equal to a new number each time. So make the value of the variable for one group equal to one, the second one equal to 2, the 3rd to 3, etc.

# Ex 2. ADDITIONAL REVIEW

***Here we would suggest taking a break, and returning to the manual another day. If you do follow our suggestion, please use the quick exercises below to review chapters 1-5 before you continue. If you continue straight from the previous exercises, it would still be useful to try your hand at these as well, as you will learn some small—but useful –commands and tricks, and generally get more practice.***

Please open Stata by double-clicking on the Stata icon. Now let’s open the original data file that we used (intro.dta), but use a different method to do it than before. This new method involves opening the file internally from Stata and will come in very handy soon.

Just to remind you, this dataset contains math and reading scores, as well as attendance rates, of students in various schools. It also has information on their gender.

Type in the following, replacing “put directory here” with the location of the intro.dta file on your computer. For example, mine is located in “C:\Users\IPA\ IPA Training\2014\_India\01\_Course Contents\Stata”. You can find the file in your USB folder, under Stata > 101 > Data. If you are not sure what the proper path is, go to the location of the file, right click the file and then select “Properties”, then the “Security” tab to find out what the directory is. Note that if the file is on your desktop, for example, you don’t just put “Desktop/intro.dta”. The proper path would be “C:\Users\IPA\Desktop\intro.dta”.

**use** “PUT YOUR DIRECTORY HERE/intro.dta”

Now let’s review some of the commands we learned last time. First, how do you find the average of the attendance rate?

A: **summ** *attendance*

How would you look at all the values that attendance rate takes and how frequently they occur?

A: **tab** *attendance*

Now suppose the kids have to re-take the year of school if they miss more than 40% of classes. What percentage of all the kids currently have attendance rates that are too low? (Tip: create a new variable for this)

A: **gen** *lowattendance* = ( *attendance* < .60 )

**tab** *lowattendance*

12.82% have attendance rates that are too low.

What is the low-attendance percentage in private versus public schools? (Tip: you don’t have to create the private school variable anew, although you can. Instead just specify which schools are private (1,3) and which are public (2,4).)

A: **tab** *lowattendance* if *school* == 1 | *school* == 3

**tab** *lowattendance* if *school* == 2 | *school* == 4

10.5% in private and 15% in public.

What is the average attendance rate for boys in public schools?

A: **sum** *attendance* if (*school* == 2 | *school* == 4) & *female* == 0

80% average attendance

# Chapter 6: DO-FILES

1. WHAT IS A DO-FILE AND WHY DO WE NEED ONE?

Let’s imagine the following situation - you just found out you have to present your results to a partner or a PI – all the averages you produced and comparisons you made. Suppose you also found out that the data you had used to produce all the averages in chapters 1-5 was not completely clean, and have only just fixed it. You now have incorrect numbers and need to re-do everything.

How would you go about it? Would you reproduce everything you did for class 1 from scratch? Can you do it? How long would it take you to do? Just re-typing all those commands into Stata in order and checking them would take an hour.

An important feature of any good research project is that the results should be reproducible. For Stata the easiest way to do this is to create a text file that lists all your commands in order, so anyone can re-run all your Stata work on a project anytime. Such text files that are produced within Stata or linked to Stata are called **do-files**, because they have an extension .do (like intro\_exercise.do). These files feed commands directly into Stata without you having to type or copy them into the command window.

An added bonus is that having do-files makes it very easy to fix your typos, re-order commands, and create more complicated chains of commands that wouldn’t work otherwise. You can now quickly reproduce your work, correct it, adjust it, and build on it.

Finally, do-files make it possible for multiple people to work on a project, which is necessary for cooperating with your PI and for handing the project off to a different PA/RA or a PC/RM.

1. STARTING A DO-FILE

To start a new do-file in Stata either:

* In the Menu bar up top, go to Window > Do-File Editor > New-Do File
* Or simply click the ‘New Do-File Editor’ button



A blank do-file will open.

Now go to your command window and highlight all the commands you used in your last exercise (do this by clicking on the 1st command, holding shift and clicking on the last command in the review window). Now copy the highlighted commands (press Ctrl + C) and paste (Ctrl + V) them into the blank do-file document.

Congratulations! You’ve now created your first do-file! That’s all a do-file is: a list of commands that you want Stata to execute and remember. Please edit your do-file by removing any commands that were typed wrong, etc, so it runs smoothly.

1. EXECUTING COMMANDS

To execute a command in a do-file, highlight it and click on the “Execute Selection (do)” button—the last button on the menu (see illustration below). The keyboard shortcut on a PC for this is Ctrl + D.



***Note:*** If you highlight anything in a line, Stata will run the whole line, not just the highlighted part. If you don’t highlight anything in the do-file and just click “execute”, Stata will run the entire file, line by line, from the beginning.

Normally you would type commands directly into the do-file instead of copy-pasting them from Stata, since it is much more efficient. So, from now on if we ask you to execute a command please type it out in the do-file and then execute it.

# Chapter 7: REPRODUCIBILITY: A QUICK GUIDE

1. FOLDER STRUCTURE AND ORGANIZATION

When you first start the project, immediately set up a folder structure for data cleaning/analysis to which you will conform. *(Don’t do this right now, but re-organize your folders later if they are not structured this way already).* In general, try to keep everything related to the data analysis of the project in one big folder and have several subfolders. Some typical subfolders that are good to have are:

* + - *Do* (for your do-files)
    - *Raw* (for your raw data)
    - *Data* (for your processed data)
    - *Backup* (for your raw data backup, plus some of the most important do-files or processed data. Keep it updated!)
      * + ***Note:*** you need to also back up your raw data elsewhere on the computer and externally, just in case

Some people also use folders like

* + - *Graphs* (for your graphic output)
    - *Tables* or *Output* (for other output)
    - *Communications* or *Discussions* (to keep track of any data cleaning/analysis-related discussions, decisions, issues)
    - *Log* (for your logs)

1. MASTER DO-FILE

Now suppose you’re pretty far into dealing with your project’s data and you have a whole bunch of do-files that perform different stages of cleaning and analysis. What’s the best way to keep them organized?

A master do-file is the main do-file for a project that calls (i.e. runs) all the other cleaning and analysis files in order. Basically, you should be able to open the master do-file and click “Execute”, and have it automatically start with the raw data, go through all the cleaning, and end up with the final output in predetermined location.

The command to embed other do-files within the master—meaning run the other do-files out of the master file—is simply:

**do** “location\_of\_do-file/do-file\_to\_execute.do”

Another thing a master do-file should do is explain the project and the data cleaning and analysis process cohesively. Once again, you want to be able to open the file and easily identify what the project is, who the person responsible is, what the important variables are, what sort of cleaning and analysis has been done, what the output is, and where to refer for further information and detail on cleaning and analysis.

You can find a sort of guide for a master do-file on Box in the “[Doing Reproducible Research](https://ipastorage.box.com/s/oo8z3gdoz9nl9vmiqp6b)” guide.

1. ANNOTATION

One of the most important things you can do for keeping your work replicable is to always, without fail, annotate your coding as you go and keep your do-files organized. This is not only useful to you (when you inevitably forget what you were trying to check when you tabulated this variable or why you needed to clean up that variable), but to all the people who will take over your project or will have to look at it later.

***What do we mean by annotating?***

It means that the do-file should be well-organized and contain lots of comments (that Stata won’t read as commands) explaining what each step of your coding does and why it does it.

For example, in your do-file (that you just created), you can explain why you included the following commands:

**gen** *lowattendance* = ( *attendance* < .60 )

**tab** *lowattendance* if *school* == 1 | *school* == 3

**tab** *lowattendance* if *school* == 2 | *school* == 4

By leaving a comment on top of them that says:

*\*Now we will create a variable to see how many people are at risk of having to retake the class due to low attendance (less than 60%), and how that percentage differs by public and private schools*

***Note*** that asterisk \* in front of the comment. Basically, a \* in front of a line tells Stata not to execute that line. This is one way to create comments. If the comment is long, however, or contains several lines, it’s better to use another commenting format:

**/\*** If *your comment* is long format it like this so that you can break

it up into two different lines

**\*/**

Stata will ignore anything in these brackets **/\* \*/**

Another way to comment is to include two forward slashes “//” after a command in a do-file. This tells Stata to only read the line until the double slash, and then skip to the next line directly, and allows you to add comments to the end of the line.

Please note that these commenting techniques only work in do-files, and will not work in the command window of Stata directly (there is no need to annotate there).

***Purpose of the do-file***

An important part of do-file annotation is the initial information about the do-file that is listed at the beginning. So, when starting a do-file, at the very top you should always create a section that contains the following information:

/\*

*Name: beginners\_class2.do*

*Date Created: January 28, 2014*

*Date Last Modified: January 29, 2014*

*Created by: Gean Spektor*

*Modified By: GS*

*Last modified by: CB*

*Uses data: intro.dta*

*Creates data: intro\_modified.dta*

*Description: This file is a part of exercises that are designed as an introduction to Stata for beginners, used at IPA-JPAL Staff Training. This particular do-file is created in order to give the trainees their first experience with a do-file and demonstrate some of the common practices for do-file organization*

\*/

1. TYPICAL OPENING COMMANDS

Another important and constant part of the do-file is that the coding should always start at least with the following set of commands:

**clear**

**set more off**

**set mem 50m**

1. CLEAR– will “clear out” any previous dataset that Stata has loaded.

If you tried to load a dataset right now within the Stata window you have open, it would give you an error, saying “no; data in memory would be lost”. This is because once you already have a dataset open and have made changes within it (even ones you don’t want to save), Stata will not open another dataset on top of it. To avoid this issue, every time you write a do-file that opens new data, start it with “clear”.

1. SET MORE OFF – Stata typically will only run enough operations to fill up the results window, and then pause until you click any key to tell it to continue (or show “more”). While this can be useful when you’re looking through results in the window for the 1st time, for the most part having to click a button several times to get Stata to execute a set of commands becomes very annoying very fast. To avoid this, you want to **set more off**, or tell Stata that you don’t want it to ask you whether you would like to see more and want all the results processed at once.
2. SET MEMORY - This command expands the amount of memory that Stata allocates to running the dataset. Memory needs to be increased any time you are running a dataset bigger than 1 mb, which is most of the time with our data. **Note**: for Stata 12 users and above, you do not need this command, since Stata now adjusts the memory allowance automatically for you to the correct level.

To determine how high to set the memory, look at the size of your dataset (right click on the dataset and go to “Properties”) and then add about 20% more memory allowance on for processing and adding new variables.

*The syntax of the command is*

**set mem** 30m

Where **set mem**  is the **set memory**  command (shortened), and **30m**  is how high you want the memory set (in this case, 30 megabytes).

1. OTHER GOOD COMMANDS TO INCLUDE IN THIS INITIAL SECTION:
2. VERSION OF STATA– when multiple people work on a do-file, they sometimes end up using different versions of Stata (in the same way someone might have Windows 7 and another person a Windows XP or a mac). IPA typically works with Stata 12 or 13. The commands in Stata are mostly standardized across versions, but occasionally the syntax changes slightly and a command won’t run properly. In order to avoid this you want to right away tell Stata which version of Stata to use, setting it to the lowest one available in the group that will use the files. The syntax is simply

**version 12.0**

1. LOG CLOSE **–** a log automatically creates a record of all the commands run and their results in a separate text file (it’s different from a do-file in that it is created by Stata by recording what goes on in its results window, including output). Logging is a great strategy for replication. For more information on logs, see the [Stata 102](https://ipastorage.box.com/stata-102) training.

The important thing to note here is that it is impossible to open a new log if one is already running. That is why you need to close a log by using the command “**log close**”. However, if there is no log open, “**log close**” will create an error, which will stop the running of your do-file.

**Cap**, or **capture**, is a prefix to commands that tells Stata to execute the command if it is correct and skip over it if there is an error.Generally it is good to be careful with this command, since it can mask serious errors, but in this case it is appropriate.

**cap log close**

Now let’s add these commands to your existing do-file, and then save the do-file in the ‘Do’ subfolder in your Stata folder on the USB. Let’s close the do-file we’ve created and open a new do-file for the next few exercises. Please start it with appropriate commenting and the typical set of commands as well.

# SUMMARY

|  |  |
| --- | --- |
| *Now you should know:*   * *What Stata is, what it looks like and how to operate it* * *How to look at data and how to create some basic statistics* * *How to use and/or in Stata correctly* * *How to save data* * *How to generate new variables, modify existing ones, and drop them* * *How to sort your data* * *What a do-file is, how to open one, how to execute commands in one* * *How to make your work reproducible*   + *What the proper folder structure is for Stata work for your project*   + *How to use a master file*   + *How to annotate your files (always do it!)* * *What are some necessary commands in a do-file* | *The commands you should’ve learned are:*  **display**  **browse**  **summarize**  **tabulate**  **list**  **if** *(condition)*  **&** *(and)*  **|** *(or)*  **save**  **generate**  **replace**  **drop**  **sort**  **do**  **clear**  **set more off**  **set mem**  **version**  **cap log close** |
|  |  |