Data Preparation in SPSS

Jamie DeCoster
Center for Advanced Study of Teaching and Learning
University of Virginia
350 Old Ivy Way
Charlottesville, VA 22903

August 15, 2012

Table of Contents

Version History

Date	Changes
2012-08-04	Created
2012-08-05	Added syntax vs interactive mode section Removed section on automatically including syntax in output Started best practices section
2012-08-06	Added file archiving sections to best practices
2012-08-07	Started working on interactive mode procedures
2012-08-08 to 2012-08-09	Continued work on interactive mode procedure
2012-08-10	Started syntax section
2012-08-13 to 2012-08-14	Continued syntax section
2012-08-14a to 2012-08-15	Did a pass through the document. 2012-08-15 is the first public version.

Interactive Mode versus Syntax Mode

There are two basic ways that you can work with SPSS. Most users typically open up an SPSS data file in the data editor, and then select items from the menus to manipulate the data or to perform statistical analyses. This is referred to as *interactive mode*, because your relationship with the program is very much like a personal interaction, with the program providing a response each time you make a selection. If you request a transformation, the data set is immediately updated. If you select an analysis, the results immediately appear in the output window.

It is also possible to work with SPSS in *syntax mode*, where the user types code in a syntax window. Once the full program is written, it is then submitted to SPSS to get the results. Working with syntax is more difficult than working with the menus, because you must learn how to write the programming code to produce the data transformations and analyses that you want. However, certain procedures and analyses are only available through the use of syntax.

Why syntax is typically better

Doing your data preparation in Syntax Mode is preferable to Interactive Mode for several reasons. First, programs can be saved and rerun at a later point in time. This means that if you ever need to make a change to the way you created your data set, you can just modify the syntax and rerun it. It will typically take much more work to undo changes you made to a data set directly. Second, once you write a syntax program to perform a task, you can often use the code again later on when you need to perform a similar task in a different circumstance. Modifications that require only small changes to a syntax program would require you to completely redo all of your work if you were making all of your changes in Interactive Mode. Third, creating syntax programs provides you with a permanent record of the data preparation steps you performed, making it easier to interpret the variables in your final data set. Fourth, using syntax makes it relatively easy to add variables into the final version of a file. This allows you to work with smaller data files because you don't need to keep extra variables "just in case" you might need them in the future. When preparing your data by hand, you will typically want to keep variables unless you know for sure that you won't need them. When using syntax, you will typically want to exclude variables unless you know for sure that you will need them.

Interactive mode is easier and generally quicker if you only need to perform a few simple transformations or analyses on your data for exploratory purposes. However, you should probably use syntax for any data preparation for analyses you would actually report in a paper or presentation.

Getting syntax out of interactive mode commands

Luckily, there are several ways to get SPSS syntax code without having to write it yourself. Whenever you make selections in interactive mode, SPSS actually writes down syntax code reflecting the menu choices you made in a "journal file." The name of this file can be found (or changed) by choosing **Edit → Options** and then selecting the **General** tab. If you ever want to see or use the code in the journal file, you can edit the journal file in a syntax window.

SPSS also provides an easy way to see the code corresponding to a particular menu function. Most selections include a **Paste** button that will open up a syntax window containing the code for the function, including the details required for any specific options that you have chosen in the menus.

Finally, you can have SPSS include the corresponding syntax in the output whenever it runs a statistical analysis. To enable this option:

- Choose **Edit → Options**.
- Go to the **Viewer** tab.
- Check the box next to **Display commands in the log** in the **Initial Output State** section in the lower-left corner.

Best Practices for Data Preparation in SPSS

Use syntax from start to finish

When using programs, we suggest that you make them as complete as possible, so that you can run them on their own without having any human steps in the process. Ideally, your data preparation syntax should start by loading the raw data file, perform all of the necessary corrections, and save the final version of the data set. There are two main advantages to making your syntax complete. First, it gives you a complete record of all of the data preparation that is needed for the analyses, providing you with a clear understanding of exactly what each of your variables represents. Second, it makes it very easy to regenerate the data set should you decide to change your data preparation steps, change the variables included in the data set, or change the cases included in the data set.

Maintain accurate codebooks for raw data sets

You should create codebooks for all raw data sets containing detailed information about each variable. Given access to the codebook, a naïve researcher should be able to analyze the data without checking an external source. If any of the variables in the data sets were collected from survey items, the codebook should report the exact questions and response options in the survey.

One way to create the codebook is to print out a copy of the survey and then write down the variable names next to each question. The final version can then be scanned and saved in the archive. You should give the codebook a filename that is similar to the data set it explains so that they are listed next to each other in the directory.

Use summary data sets for analysis

It can be very difficult to find specific variables in the raw data sets if a study used a large number of measures. Assuming each scale item was recorded as its own variable, a moderately-sized survey can easily result in a data set containing hundreds of variables. Navigating a data set of this size is a cumbersome task, even with an accurate codebook. As a solution to this problem, you should consider creating summary data sets that only contain the variables needed for a particular analysis. For example, most investigators do not analyze the responses to individual trials or items, but instead only look at composite measures. A summary data set would contain these composites, but would drop all of the variables corresponding to specific items or assessment trials. As another example, many grants collect data on a number of different issues, so the final data sets from the project may have a large number of variables, even after creating composites. Researchers investigating a specific set of hypotheses for a paper might choose to create a summary data set that only includes the variables needed for their tests. Summary data sets should be saved as separate files, because it is important to retain the raw data sets in case you want to recalculate a composite measure.

Preserve the original raw data sets

The raw data from a study are typically modified multiple times before they are subjected to analysis. These modifications may remove invalid cases, transform or recode existing variables, or create new composite scores. Each time a change is made to a data set, however, there is a chance that an error may accidentally be introduced. It is therefore best to save a copy of the original raw data file without any modifications so that the data sets can be corrected in case an error is found.

Create data dictionaries for summary data sets

Data dictionaries are to summary data sets as codebooks are to raw data sets. Data dictionaries should focus on the information an investigator would need to accurately analyze the data set, and are most helpful when more than one person will be working with a summary data set. If the person who originally wrote the program to create the data set is the only person who will be analyzing the data set, it is probably not necessary to create a data dictionary since that person will likely be able to read their own syntax program to understand the contents of the data set. However, if other researchers will be using the data set or the researcher is not using a program to create the summary data set, it is worth the time to create a data dictionary.

The data dictionary should provide the name of the data set to which it refers, as well as a short description of the study from which the data were collected. The remainder of the dictionary describes the variables in the data set. For each variable, the data dictionary should report its name and provide an explanation of the values it can take. For categorical variables, this requires that you report which values correspond to each possible category. For variables derived from scales, this requires that you report the range of possible values, and what high and low values mean theoretically. For naturally continuous variables, this requires you to report the unit of measurement. Every variable in the data file should be represented in the data dictionary; however, sets of similar variables (like measures of the same variable at different time points) can be described together in a single entry to save space.

Naming variables

Use descriptive names for variables

Using descriptive variable names makes it easier to select which variables to use in an analysis and also makes it easier to read the statistical output. Consider using variable labels and value labels to help keep track of the real meaning behind your variables.

Coordinate variable names across data sets you may eventually merge together

When preparing the data for a study that makes use of multiple data files, you should try to name your variables in a way that makes it easiest to combine the files. If you will be merging data sets that have information on the same measures but different subjects (such as data sets from the same study conducted at different sites), then you should be sure that the data sets all use the same names for any variables they have in common. If you will be merging data sets that have information on the same subjects but represent different measurements (such as data sets representing assessments made on the same subjects at multiple time points), you will need an identification variable (such as the subject number) that uniquely identifies each subject. The identification variable must appear in each data set and have the same name across all of the data sets so that observations from the same subject can be linked together. Other variables should have unique names so that they can be added to the merged data set without conflicting with each other. You cannot have two variables with the same name in the same data set, so merging will be easiest if the nonidentifier variable names are all unique. If the data sets being merged represent the same assessments over time, you might consider adding a suffix to the variable names indicating the time point to make them unique (e.g., MMSE_T1).

Suggested settings

There are several settings that you can choose that will make data preparation in SPSS easier.

Using only one data set at a time

By default, SPSS opens up a new data window every time that you load a new data set. This can be useful if you are manually comparing two data sets, but can become confusing if you are unintentionally opening many data sets at a time. It can also be problematic if you are using programs to perform data preparation. SPSS will not allow you to overwrite a dataset that is already open in a different window. If you have a program that is designed to overwrite an existing data file and that file happens to be open in another window, SPSS will automatically change the name of the file when you try to save it. This can cause problems if you have other programs that are supposed to access the updated file because those programs will instead load the older file.

To avoid these problems, you can tell SPSS that you only want to open one data set at a time. When you do this, if you already have a data set open and you issue a command to load a new data set, SPSS will automatically close the old data set and replace it with the new one. It will also change the syntax that is pasted from the menus so that you are always working with a single dataset.

To tell SPSS to only use one data set at a time:

- Choose **Edit → Options**.
- Go to the **General** tab.
- Check the box next to **Open only one dataset at a time** in the **Windows** section in the lower-left corner.

Display names in variable lists

SPSS has the ability to assign labels to variables as well as to specific values that variables can have. When you set up these labels, SPSS will then use them in place of the variable names when you are choosing variables for your analyses. Sometimes, however, the variable names are very long, so that you cannot see the whole name in the limited space you have in the variable selection window. This can be problematic if you have several items that only differ from each other in that part of the label that you can't see. To prevent this from becoming a problem, you can tell SPSS to present the actual variable name instead of the variable label when it presents a variable selection window.

To tell SPSS to use variable names instead of labels in its variable selection windows:

- Choose **Edit → Options**.
- Go to the **General** tab.
- Check the box next to **Variable names** in the **Variable Lists** section in the upper-left corner.

Customizing the syntax editor

There are several things you can do to change the way that the syntax window looks and operates. Here are some settings that might make things easier for you.

Change the font size

The default font size is pretty small. You might want to make it larger so that it is easier for you to read your programs.

To change the font size of the syntax editor:

- Open a syntax window.
- Choose **View → Fonts**.
- Select the font characteristics you want for the editor.

Turn off the navigation pane

The navigation pane is designed to give you an outline of the different sections of your program. However, it doesn't do a good job since it basically is just a list of every command in your program. Instead of having the navigation pane, you might decide that you'd like to have extra space for the editor itself so that you can see more without scrolling.

To turn off the SPSS syntax navigation pane:

- Choose **Edit → Options**.
- Go to the **Syntax Editor** tab.
- Uncheck the box next to **Display the navigation pane** in the **Panes** section in the lower-right corner.

Turn off auto complete

SPSS has a feature where it will automatically show a pop-up window as you are typing in the syntax editor that has a list of commands that are consistent with the letters you type on a line. This can be helpful if you commonly use the mouse when you are editing, but it can be annoying if you typically rely on keyboard shortcuts because it displays the pop-up window over part of your program. If you do not find yourself using the auto-complete pop-up window, you might consider turning it off.

To turn off the auto-complete pop-up window:

- Choose **Edit → Options**.
- Go to the **Syntax Editor** tab.
- Uncheck the box next to **Automatically display the auto-complete control** in the **Auto-Complete Settings** section in the upper-right corner.

Using SPSS Interactive Mode

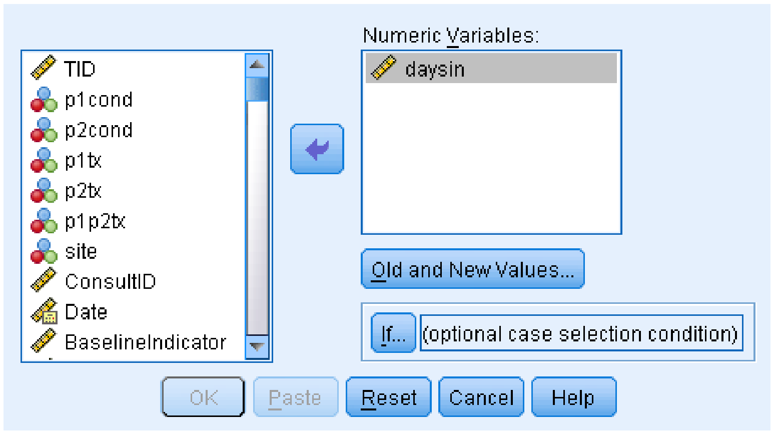
Transform → Recode into Same Variables

This procedure is used to create a categorical variable based on the values of another variable in the data set. It is the best option when you want to create a categorical distinction based on an existing numeric variable (such as a median split), when you want to combine some of the categories in an existing categorical variable, or when you simply want to change the values assigned to an existing categorical variable. This procedure is specifically used to replace the values of one variable with a new set of categories. When you do this, the original data is removed from the current version of the data set. If you want to leave the original data in the data set and put the new categories into a new variable, see the **Transform → Recode into Different Variables** procedure described on page 13.

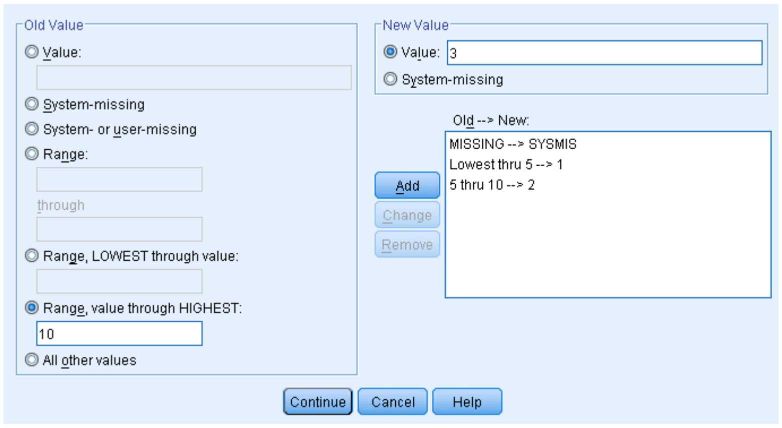
It is generally recommended that you use numbers to code different levels of your categorical variables in SPSS. Although SPSS variables can have letters or words as values (these types of variables are called "string" variables), there are several analyses that can only work with numeric variables, even when analyzing categorical variables.

To recode into the same variable:

- Choose **Transform → Recode into Same Variables**. This will open a new window that looks like the following image.



- Select the variable in the left box that you want to use to determine the values of the new categorical variable and then press the arrow button. This will move the variable to the **Variables** box. If you want to use the same rules to create multiple categorical variables, you can move multiple variables from the left box to the **Variables** box.
- If you only want to apply the recoding to a subset of cases that meet a criterion condition, you can define this condition by clicking the **If** button. For more information on using the **If** button, see page 18.
- Click the **Old and New Values** to define the coding scheme. This will open a window that looks like the following figure.


Old Value

Value:

System-missing

System- or user-missing

Range:

through

Range, LOWEST through value:

[✓ Range, value through HIGHEST:] 10

All other values

New Value

[✓ Value:] 3

System-missing

Old --> New:

```
MISSING --> SYSMIS
Lowest thru 5 --> 1
5 thru 10 --> 2
```

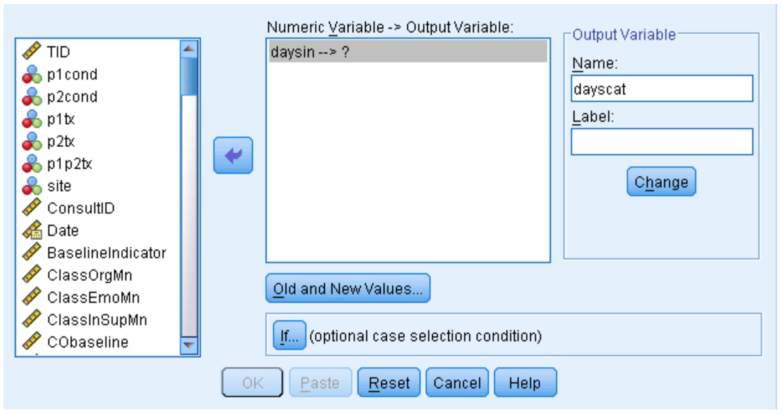
- Here you define how the values of the original variable are transformed into the values of the new variable. On the left side you identify values of the original variable, and on the right side you indicate what values those cases should have in the new variable. Each time you define an old value/new value pair, you click the **Add** button and it will add the pairing to the **Old - -> New** box. You can identify pairings individually, or you can assign ranges of old values to a single new value.
- When you have set up all of the pairings, click the **Continue** button.
- Click **Paste** if you want to save the syntax for the recode, or click **OK** to run it immediately.

Transform → Recode into Different Variables

This procedure does the same thing as **Transform → Recode into Same Variables**, except that instead of overwriting the original variable, the new, recoded variable is added to the data set under a different name. Most of the way you would use this procedure would be the same as **Transform → Recode into Same Variables**, but the initial screen is a little different to give you the opportunity to define the new variable name.

To recode into a different variable:

- Select **Transform → Recode into Same Variables**. This will open a window that looks like the following figure.


Numeric Variable -> Output Variable:

daysin --> ?

Output Variable

Name: dayscat

Label:

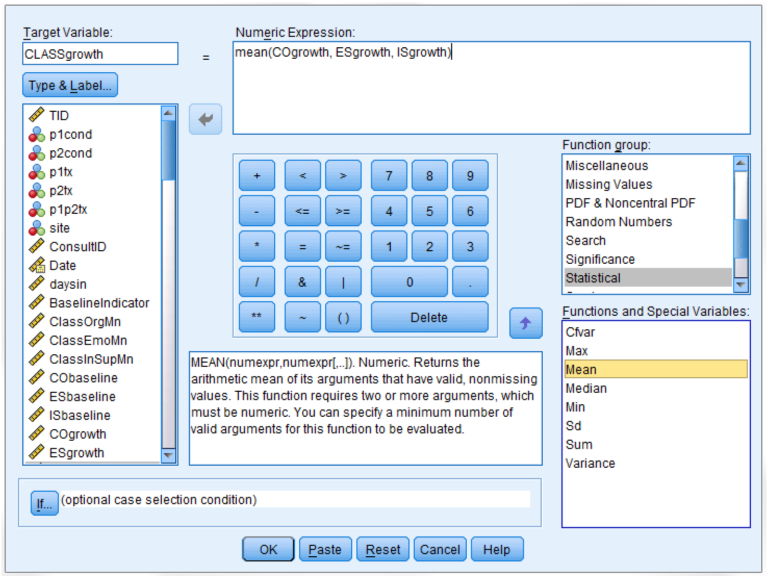
- Select the variable in the left box that you want to use to determine the values of the new categorical variable and then press the arrow button. This will move the variable to the left side of the **Numeric Variable - -> Output Variable** box.
- You then need to type in what you want the name of the new recoded variable to be in the **Name** box in the **Output Variable** section.
- Finally, you click **Change**, which will move the new name to the right side of the **Numeric Variable - -> Output Variable** box. If you want to apply the same recoding to multiple variables, you can keep on adding multiple pairs of variable names to the **Numeric Variable - -> Output Variable** box.
- After this you click the **Old and New Values** to define the coding scheme. This works exactly the same way as described in the **Transform → Recode into Same Variables** section that begins on page 12. See that section for more details.
- After defining the coding scheme, click **Paste** if you want to save the syntax for the computation, or click **OK** to run the recode.

Transform → Compute Variable

This procedure is used to create one variable as a mathematical function of other variables in the data set. It also has the ability to generate random numbers from a number of different distributions. It is the most common way to transform one numeric variable into a different numeric variable.

To perform a computation:

- Select **Transform → Compute Variable**. This will open a new window that looks like the following image.



Target Variable: CLASSgrowth = Numeric Expression: mean(COgrowth, ESgrowth, ISgrowth)

MEAN(numexpr,numexpr[,..]). Numeric. Returns the arithmetic mean of its arguments that have valid, nonmissing values. This function requires two or more arguments, which must be numeric. You can specify a minimum number of valid arguments for this function to be evaluated.

- Type the name of the variable you want to create in the **Target Variable** box. If this is the same name as an existing variable, you will overwrite the contents of that variable with your computation.
- Type the formula you want to use for the computation in the **Numeric Expression** box. You can use the variable list on the left or the calculator buttons in the middle of the window when creating your formula, but you don't have to – you can just type your formula directly into the box if you want.
- The **Function group** and **Functions and Special Variables** boxes can be used to see lists of the different functions that SPSS has available. You can double-click a function in the Functions and Special Variables box to add it to your formula, or you can just type it in manually.
- Click **Paste** if you want to save the syntax for the computation, or click **OK** to run the computation.

This procedure duplicates the function of the **compute** statement in SPSS syntax, which is described in detail on page 27.

Arithmetic operations

All of your standard arithmetic operations are available for use with **Transform → Compute**. Here is a list of the most common operators.

- + (addition)
- - (subtraction)
- * (multiplication)
- / (division)
- ** (raising something to a power)

Order of operations

For many mathematical and statistical formulas, you can often get different results depending on the order in which you decide to perform operations inside the formula. Consider the following equation.

X = 12 + 6 / 3

If we decide to do the addition first (12 + 6 = 18; 18 / 3 = 6) we get a different result than if we decide to do the division first (6 / 3 = 2; 12 + 2 = 14). Mathematicians have therefore developed a specific order in which you are supposed to perform the calculations found in an equation. According to this, you should perform your operations in the following order.

1. Resolve anything within parentheses.
2. Resolve any functions, such as squares or square roots.
3. Resolve any multiplication or division operations.
4. Resolve any addition or subtraction operations.

This would mean that in equation 1.1 above, we would perform the division before we would do the addition, giving us a result of 14. If this does not match what you want, you can always add in parentheses to make it clear what operations you want to perform first.

Variable lists

A number of functions, such as **mean** and **sum**, are designed to perform a calculation based on the values of multiple variables. There are two basic ways that you can provide these variable lists to the functions. First, you can name each variable individually, separated by commas. For example, to take the mean of the variables **var1**, **var2**, **var3**, **var4**, and **var5**, you could use the following formula:

```
mean(var1, var2, var3, var4, var5)
```

This method, however, can become cumbersome if you have a large number of variables. SPSS therefore gives you a shortcut to refer to an entire set of variables, as long as the variables are all next to each other in the data set. In this case, you can refer to the entire collection of variables by giving the name of the first variable in the list, followed by the word **to**, followed by the last variable in the list. For example, if the above five variables were all next to each other in the data set, you could use the formula

```
mean(var1 to var5)
```

to perform the same calculation as above.

Specific compute functions

Below are explanations of the functions that are used most often as part of compute statements.

mean(variable list)

This function gives you the average of a list of variables. If any of the variables have missing values, it gives you the mean of the ones that are not missing. In many cases this will be a better result than calculating the average by hand, which will give you a missing variable as a result if any of the variables you are averaging are missing.

sum(variable list)

This function gives you the sum of a list of variables or numbers. If any of the variables have missing values, it gives you the sum of the ones that are not missing.

median(variable list)

This function gives you the median of a list of variables or numbers. If any of the variables have missing values, it gives you the median of the ones that are not missing.

sd(variable list)

This function gives you the standard deviation of a list of variables or numbers. If any of the variables have missing values, it gives you the standard deviation of the ones that are not missing.

variance(variable list)

This function gives you the variance of a list of variables or numbers. If any of the variables have missing values, it gives you the variance of the ones that are not missing.

rnd(variable)

This function rounds a number with decimals to the nearest integer. If the decimal is equal to exactly .5, then it rounds it to the number that is furthest away from zero. For example, `rnd(1.5)` would be equal to 2, and `rnd(-1.5)` would be equal to -2.

sqrt(variable)

This function gives you the square root of a number.

ln(variable)

This function gives you the natural logarithm (base *e*) of a number.

lg10(variable)

This function gives you the base 10 logarithm of a number.

exp(variable)

This function gives you the exponential function of a number (i.e., *e* raised to the number).

Reverse coding using compute

Reverse coding is a procedure where some questions in a survey are worded such that high values of a theoretical construct are reflected by high scores on the item, while other questions are worded such that high values of the same construct are reflected by low scores on the item. Researchers do this to encourage respondents to actually pay attention to the questions they are reading. Unfortunately, this means that you can't determine the overall score for the scale simply by averaging the items. Instead you must first transform the items so that they are all oriented in the same direction. For example, you might want to transform the values so that large values indicate more of the construct for all of the items. To do this, you would want to reverse the coding of the items where small values indicated a greater amount of the construct. So, if the questions in the scale had values of 1 to 7, you would reverse code an item by changing its values in the following way:

Old Value	New value
1	7
2	6
3	5
4	4
5	3
6	2
7	1

While it would be possible to perform this transformation using the **Recode** procedure, there is a simple formula you can use to do the same thing using the **Compute** procedure. The formula is

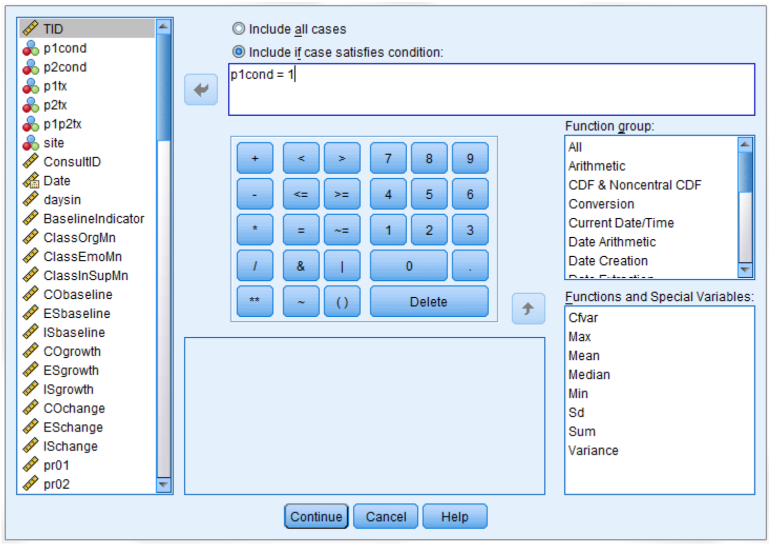
new value = (scale minimum + scale maximum) – old value

In the current example, the scale minimum is 1 and the scale maximum is 7. Therefore, the formula we'd need to use would be **8 – old value**. You can verify for yourself that this will produce the transformation described above. The formula will work for any possible scale minima and maxima, even if the scale has values less than zero.

Applying a transformation to a select set of cases

When using the **Compute Variable** selection, you can have SPSS only apply the computation to cases that meet specific criteria. To do this, you would take the following steps.

- Open up a compute window and type in the **Target Variable** and **Numeric Expression** for the computation as described above.
- Click the **If** button in the lower-left corner. This will open a new window that looks like the following image.



( ) Include all cases
[✓ Include if case satisfies condition:]

p1cond = 1

- Click the button next to **Include if case satisfies condition:**
- Type in the equality or inequality that has to be met for the computation to be performed in the box below the button.
- Click the **Continue** button.
- Click **Paste** if you want to save the syntax for the computation, or click **OK** to run the computation.

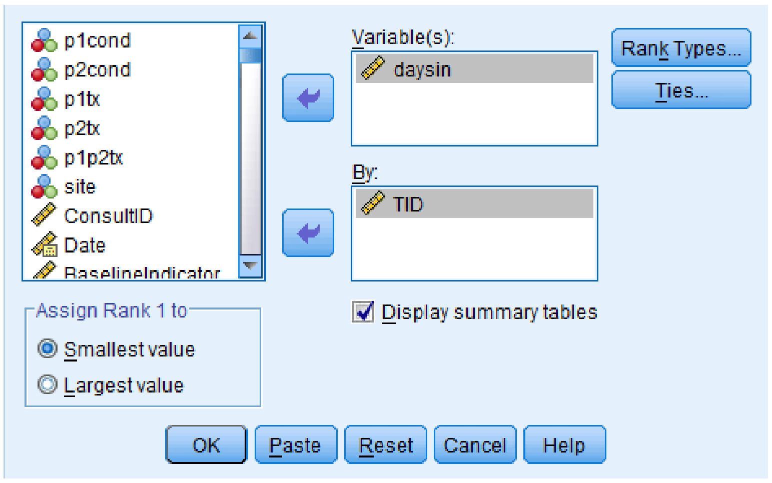
If you add this type of a condition to your computation, SPSS will only change the cases that satisfy the condition. If you created a new variable, the cells for all cases not meeting the condition are left as missing. If you overwrote an existing variable, the cells for all cases not meeting the condition are left unchanged. This allows you to apply different computations for different groups in the same variable, if you were to use this procedure multiple times with different conditions each time.

Transform → Rank Cases

This procedure will automatically generate ranks based on the values of a numeric variable. You can either create a single ranking for all of the cases in the data set, or you can have the ranking performed separately within each level of a grouping variable. For example, you could have it separately rank the scores for students within each school.

To automatically rank cases:

- Choose **Transform → Rank Cases**. This will open a window that looks like the following image.



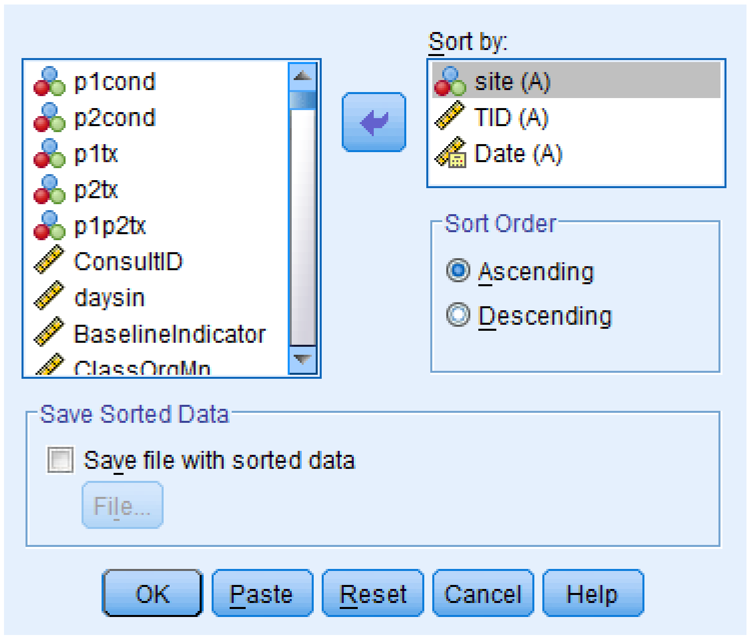
- Move the variable that is to be used as the basis of the ranks to the **Variable(s):** box. If you select more than one variable, then separate rank variables will be created for each variable you select.
- If you want the ranks to be separated by a grouping variable, move the grouping variable to the **By:** box. If you want the ranks to be calculated over the entire data set, just leave the **By:** box empty.
- By default the case with the smallest value will be assigned the rank of 1 and cases with larger values will have will have higher ranks. If you would instead prefer that the rank of 1 to be assigned to the largest value and cases with smaller values would have higher ranks, you can select the radio button next to **Largest value** in the **Assign Rank 1** to box in the lower-left corner.
- Click **Paste** if you want to save the syntax for the ranking, or click **OK** to run it immediately.

Data → Sort Cases

This procedure sorts the cases in the data set. Certain procedures, such as merging data sets, require that the cases be sorted in a specific way before they can be performed. It can also sometimes also be easier to examine the data in your data set after it has been sorted in a particular way.

To sort your cases:

- Choose **Data → Sort Cases**. This will open up a window that looks like the following image.



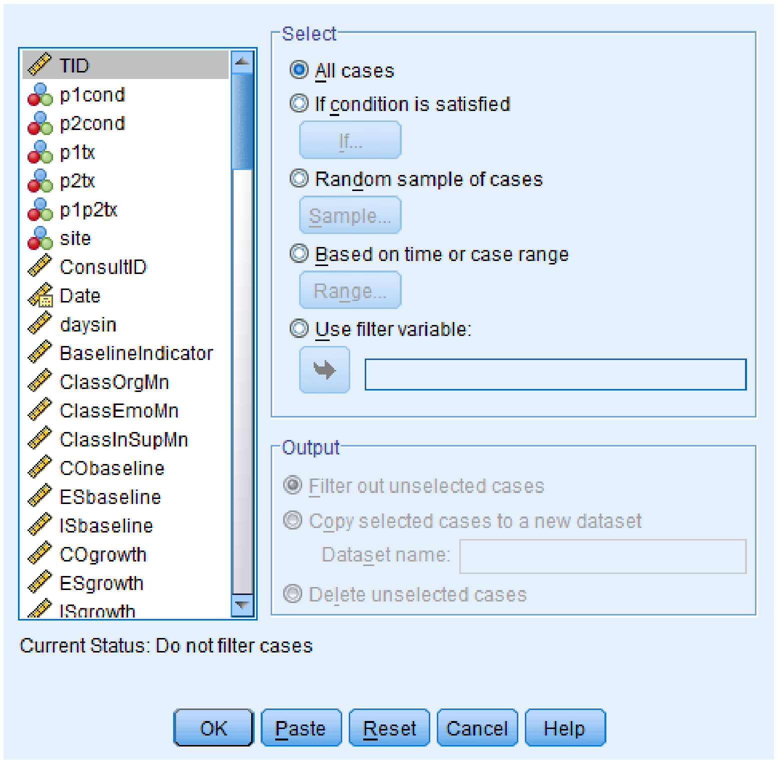
- Move the variables you want to sort by to the **Sort by:** box. You can specify a number of different variables. If you choose more than one variable, SPSS will first sort the cases by the first variable in the list. For cases with the same value on the first variable, it will then sort them based on the second variable in the list. For cases with the same value on the first and second variables, it will then sort them based on the third variable in the list. This continues through all of the variables you select.
- For each variable in the list, you can choose whether you want the cases sorted in ascending or descending order. By default, SPSS assumes that you want the cases sorted in ascending order. If you want to change this, all you need to do is click the appropriate variable in the **Sort by:** box and then click the radio button next to **Descending** in the **Sort Order** box.
- Click **Paste** if you want to save the syntax for the sort, or click **OK** to run it immediately.

Data → Select Cases

This procedure is used to remove specific types of cases from a data set. It can also be used to just temporarily remove cases from a set of analyses, but in data preparation, it is primarily used to permanently remove cases from the data set. You should be careful not to overwrite the original data file when you use this procedure to permanently remove cases from the data set in case you change your mind and you want to get the original cases back.

To remove specified cases from your data set:

- Choose **Data → Select Cases**. This will open a window that looks like the following image.

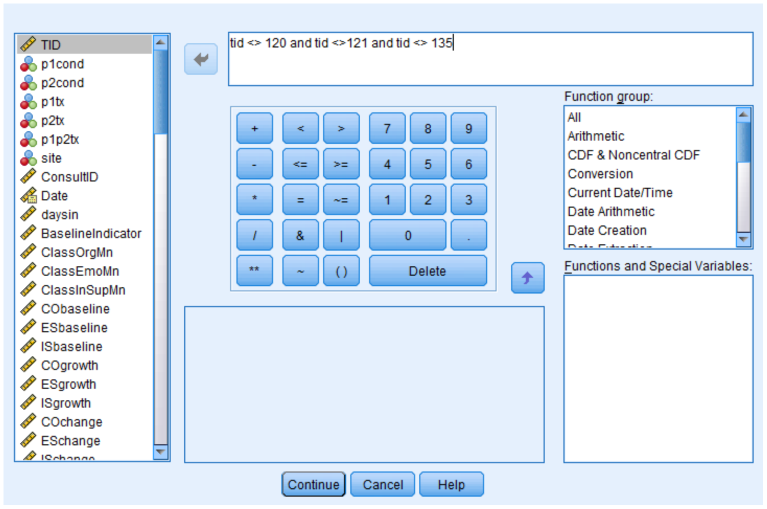


Select: [✓ All cases] If condition is satisfied / Random sample of cases / Based on time or case range / Use filter variable:

Output: [✓ Filter out unselected cases] Copy selected cases to a new dataset / Delete unselected cases

Current Status: Do not filter cases

- Next you will want to define what cases you want to keep in your data set. To do this you will want to click the radio button next to **If condition is satisfied** and then click the **If** button. This will open up a new window that looks like the following image.



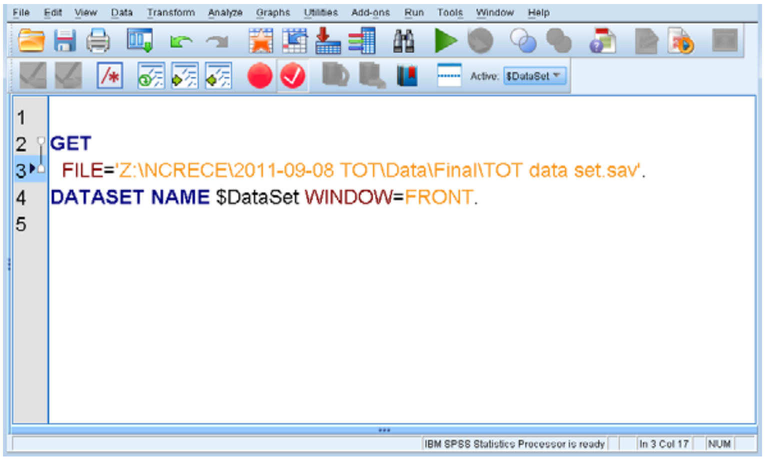
```
tid <> 120 and tid <>121 and tid <> 135
```

- You will want to select the critera for the cases you want to **keep** in the data set in the box at the top of the new window. If you want to exclude specific cases, you can simply have your condition be that a variable not be equal (represented by either <> or != in SPSS) to particular values, as illustrated in the image above.
- Once you have defined the condition for the cases you want to keep in the data set, click the **Continue** button.

- By default, SPSS will not actually remove the cases from the data set. Instead, it will simply mark them to indicate that they should not be used in analyses. If you want to actually remove the unselected cases, you need to click the radio button next to **Delete unselected cases** in the **Output** box at the bottom of the first window.
- Click **Paste** if you want to save the syntax for the selection, or click **OK** to run it immediately.

Using SPSS Syntax

To run commands using SPSS syntax, you must first either type or paste the commands into an SPSS syntax window. If you want to open a new syntax window, you can choose **File → New → Syntax**. If you do not have a syntax window but ask SPSS to paste the commands from an Interactive Mode procedure, it will automatically open up a syntax window when pasting the syntax. An example of a syntax window with some code that would open a data file is provided in the image below. Your own syntax window may look a little different depending on what options you have selected.


File Edit View Data Transform Analyze Graphs Utilities Add-ons Run Tools Window Help
Active: $DataSet

```

GET
  FILE='Z:\NCRECE\2011-09-08 TOT\Data\Final\TOT data set.sav'.
DATASET NAME $DataSet WINDOW=FRONT.

```

IBM SPSS Statistics Processor is ready | In 3 Col 17 | NUM

One thing you will notice is that every line of SPSS syntax ends with a period. If you forget to add a period, the program will report a syntax error and fail to run. You will also get syntax errors if you type in a command incorrectly.

After you generate syntax, you have to explicitly tell SPSS to run it in order for it to do anything. There are several different ways to tell SPSS to run syntax.

To run all of the syntax in a window:

- Press Ctrl-A to select everything in the window.
- Press Ctrl-R or click the green arrow button to run the syntax.

To run a section of the syntax program:

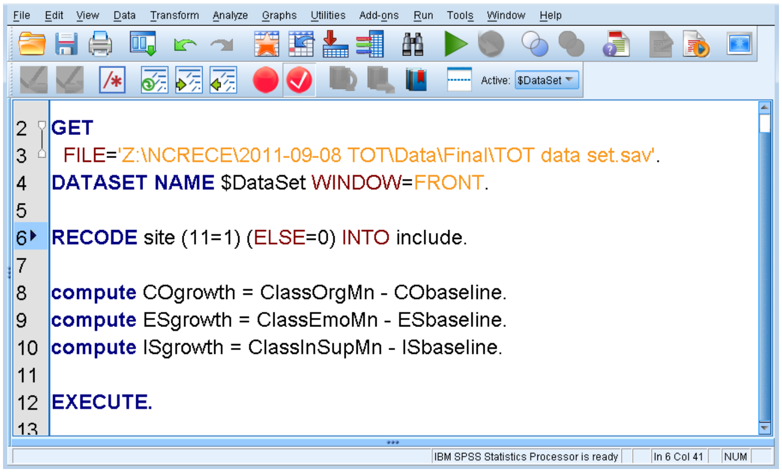
- Use either the mouse or the keyboard to select the portion of the syntax you want to run.
- Press Ctrl-R or click the green arrow button to run the syntax.

To run a single syntax command:

- Position the cursor on the line with the command. You don't actually have to select it – just put the cursor on the line.
- Press Ctrl-R or click the green arrow button to run the syntax.

execute

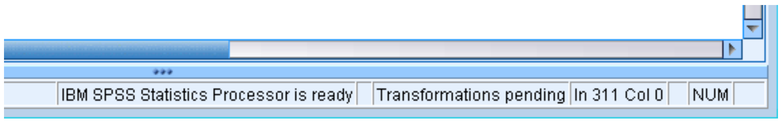
Whenever you create syntax to perform data preparation, you have to have an **execute** statement at the very end in order for your transformations to take effect. In order to conserve processing time, SPSS stores up your transformation commands in memory and only runs them when it either receives an **execute** command or you perform an actual analysis. You don't need to have an **execute** statement after every transformation – you just need to have one at the very end. Below is an example of a program with an **execute** statement at the end.



```
GET
  FILE='Z:\NCRECE\2011-09-08 TOT\Data\Final\TOT data set.sav'.
DATASET NAME $DataSet WINDOW=FRONT.

RECODE site (11=1) (ELSE=0) INTO include.

compute COgrowth = ClassOrgMn - CObaseline.
compute ESgrowth = ClassEmoMn - ESbaseline.
compute ISgrowth = ClassInSupMn - ISbaseline.

EXECUTE.
```

IBM SPSS Statistics Processor is ready | In 6 Col 41 | NUM

If you run some transformations but forget to include an **execute** statement, the transformations will be stored as "pending" and will not appear in the data set. In this case, the a message will appear at the bottom right of the syntax window saying that transformations are pending, shown in the image below.

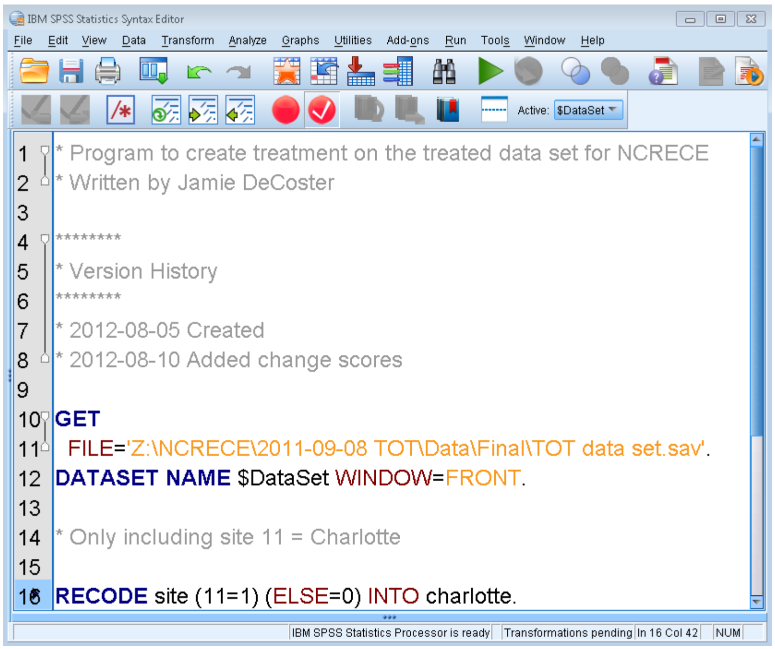


IBM SPSS Statistics Processor is ready | Transformations pending | In 311 Col 0 | NUM

If this happens, all you need to do is run a single **execute** statement and the transformations will be applied to the data set.

Comments

If you ever want to add a note to your program, all you need to do is put an asterisk (*) on the first line. SPSS will then ignore anything you type on that line and any future lines until the program either has a blank line or a period. These are called program comments. It is useful to include comments at the top of a syntax program to explain what the purpose of the program is, and it is also useful to include comments inside the syntax itself to explain what each individual section of code is doing. Below is an example of some syntax with comments in it.

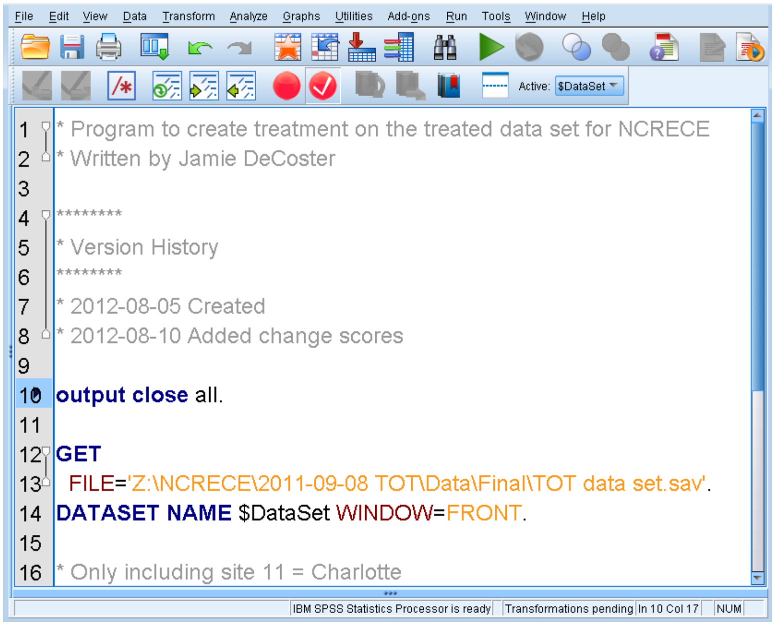

IBM SPSS Statistics Syntax Editor
File Edit View Data Transform Analyze Graphs Utilities Add-ons Run Tools Window Help
Active: $DataSet
IBM SPSS Statistics Processor is ready | Transformations pending | In 16 Col 42 | NUM

```
* Program to create treatment on the treated data set for NCRECE
* Written by Jamie DeCoster

********
* Version History
********
* 2012-08-05 Created
* 2012-08-10 Added change scores

GET
  FILE='Z:\NCRECE\2011-09-08 TOT\Data\Final\TOT data set.sav'.
DATASET NAME $DataSet WINDOW=FRONT.

* Only including site 11 = Charlotte

RECODE site (11=1) (ELSE=0) INTO charlotte.
```

One good way to use comments is to keep track of changes to your program in a "Version History" at the top of the document. That way you know what changes were made to each version of your program in case you ever want to go back and undo a change.

output close all

It is useful to include an **output close all** statement at the top of your program. This will clear out the contents of the output window that were created by prior syntax. It can sometimes be confusinsg when you are initially writing a syntax program and are correcting errors if you leave the old output because you can't tell whether the errors listed in the output have already been corrected or not. If you add an **output close all** statement at the top of the program, you know that anything appearing in the output window is coming from the program you just ran. Below is an example of a program using an **output close all** statement.

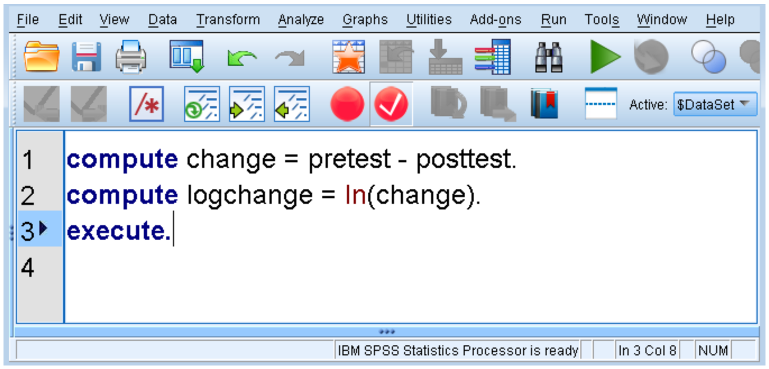

File Edit View Data Transform Analyze Graphs Utilities Add-ons Run Tools Window Help
Active: $DataSet

```
* Program to create treatment on the treated data set for NCRECE
* Written by Jamie DeCoster

********
* Version History
********
* 2012-08-05 Created
* 2012-08-10 Added change scores

output close all.

GET
  FILE='Z:\NCRECE\2011-09-08 TOT\Data\Final\TOT data set.sav'.
DATASET NAME $DataSet WINDOW=FRONT.

* Only including site 11 = Charlotte
```

IBM SPSS Statistics Processor is ready
Transformations pending
In 10 Col 17
NUM

compute

The **compute** statement is used to calculate a new numeric variable as a function of other numeric variables in your data set. It also has the ability to generate random numbers. Below is an example of a program using the **compute** statement.


File Edit View Data Transform Analyze Graphs Utilities Add-ons Run Tools Window Help
Active: $DataSet

```
compute change = pretest - posttest.
compute logchange = ln(change).
execute.
```

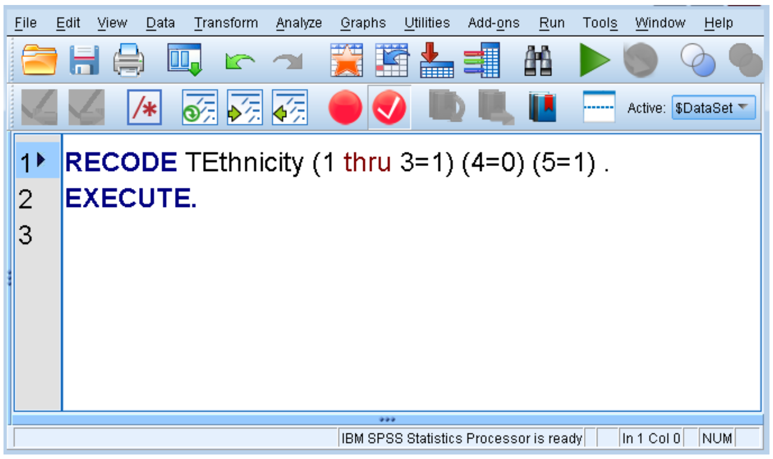
IBM SPSS Statistics Processor is ready
In 3 Col 8
NUM

The **compute** syntax command does the same thing as the Interactive Mode procedure **Transform → Compute** that was described on page 14. See that section for more information about the order of operations, the use of variable lists, and about specific compute functions.

recode

The **recode** statement is used to create a new categorical variable based on the values of other variables in the data set. The other variables can be either categorical or numeric, and the new categorical variable can either replace an existing variable in the data set or it can be assigned a new variable name.

Below is an example of syntax that overwrites the values of an existing categorical variable with new, recoded values.

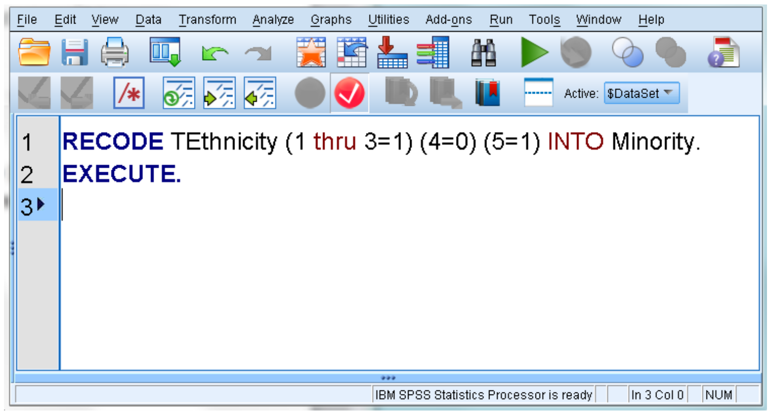

File Edit View Data Transform Analyze Graphs Utilities Add-ons Run Tools Window Help
Active: $DataSet

```
RECODE TEthnicity (1 thru 3=1) (4=0) (5=1) .
EXECUTE.
```

IBM SPSS Statistics Processor is ready
In 1 Col 0
NUM

This takes an existing ethinicity variable where all the ethnicities are coded individually and turns it into one with just two values, where 0 represents whites and 1 represents non-whites. This syntax replaces the existing values of **TEthnicity** with the new values.

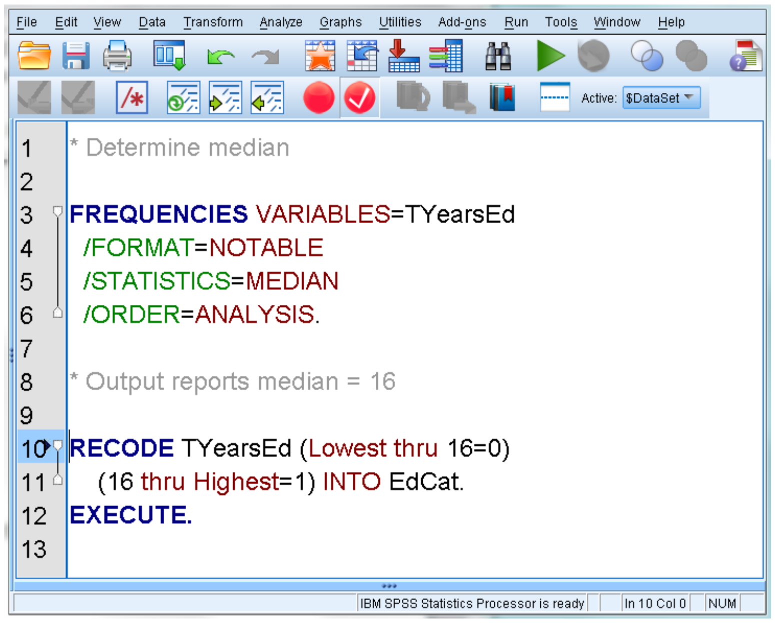
Below is an example of syntax that does the same thing, but which assigns the new categorization to a different variable instead of replacing the original variable.


File Edit View Data Transform Analyze Graphs Utilities Add-ons Run Tools Window Help
Active: $DataSet

```
RECODE TEthnicity (1 thru 3=1) (4=0) (5=1) INTO Minority.
EXECUTE.
```

IBM SPSS Statistics Processor is ready
In 3 Col 0
NUM

Below is an example of syntax that creates a median split based on the values of a numeric variable, saving the median split under a new variable name.


File Edit View Data Transform Analyze Graphs Utilities Add-ons Run Tools Window Help
Active: $DataSet
IBM SPSS Statistics Processor is ready
In 10 Col 0
NUM

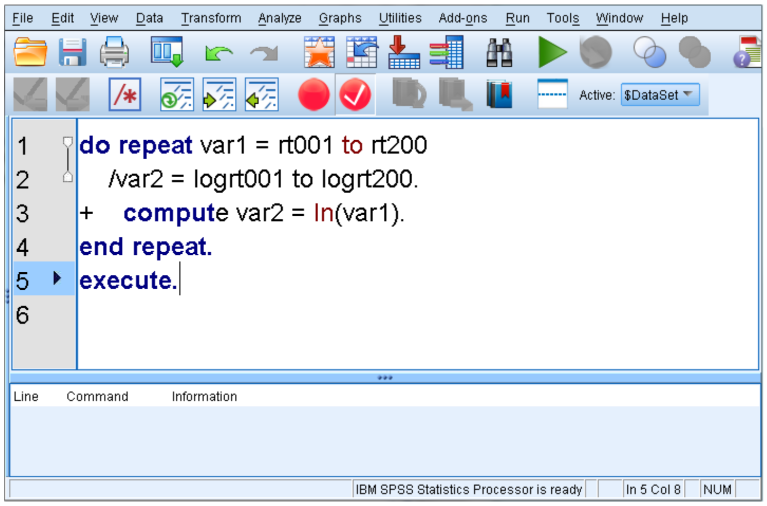
```
* Determine median

FREQUENCIES VARIABLES=TYearsEd
  /FORMAT=NOTABLE
  /STATISTICS=MEDIAN
  /ORDER=ANALYSIS.

* Output reports median = 16

RECODE TYearsEd (Lowest thru 16=0)
    (16 thru Highest=1) INTO EdCat.
EXECUTE.
```

The **recode** syntax command does the same thing as the two Interactive Mode procedures **Transform → Recode into Same Variables** described on page 12 and **Transform → Recode into Different Variables** described on page 13.

do repeat/end repeat

The combination of **do repeat** and **end repeat** statements is typically used in conjunction with either **compute** or **recode** statements to perform the same transformation on a collection of related variables. What it basically does is allow you to define a basic set of transformations, and then loop through those statements once for each variable you want to transform. This can save a large amount of work when you are performing the same transformation multiple times.

For example, let us say that you performed a reaction-time study where you recorded how long it took people to determine whether a given set of letters made up a word. By examining the average reaction time over a large number of trials, you can determine how different features of the environment might affect decision making. However, human reaction times are well known to follow a right-skewed distribution (where most of the reaction times are short, but it is possible to have some that take much longer), and so it's typical to take the logarithm of all of the reaction times before averaging them. If the study had 200 trials, this would require 200 different compute statements to transform all of the reaction times. However, the following program would do all 200 transformations using a small amount of syntax.



```
do repeat var1 = rt001 to rt200
  /var2 = logrt001 to logrt200.
+   compute var2 = ln(var1).
end repeat.
execute.
```

Lines 1 and 2 of the program define the variables that will be involved in the transformation. The program has a list of 200 reaction time variables (**rt001** to **rt200**), all of which have been associated with the name **var1**, and a set of 200 logarithm variables (**logrt001** to **logrt200**), all of which have been associated with the name **var2**. Notice that the two lists have the same number of variables – all of your lists that are created as part of a **do repeat** statement always have to have the same number of variables in them.

Line 3 of the program is a **compute** statement, which says that **var2** is to be calculated as the natural logarithm of **var1**. Since this line is between the **do repeat** and the **end repeat** statements, SPSS is going to run it multiple times, once for each entry in the lists defined as part of the **do repeat** statement. The plus sign is actually not important – it's just used to tell SPSS that the real command is going to start later on the line. This allows us to indent the **compute** statement so that we can see that it is in between the **do repeat** and **end repeat** statements and that it will be part of the loop.

In this example, the first time through the loop, SPSS replaces var1 with rt001 and var2 with logrt001, so the statement becomes

```
compute logrt001 = ln(rt001).
```

The second time through it replaces var1 with rt002 and var2 with logrt002, so the statement becomes

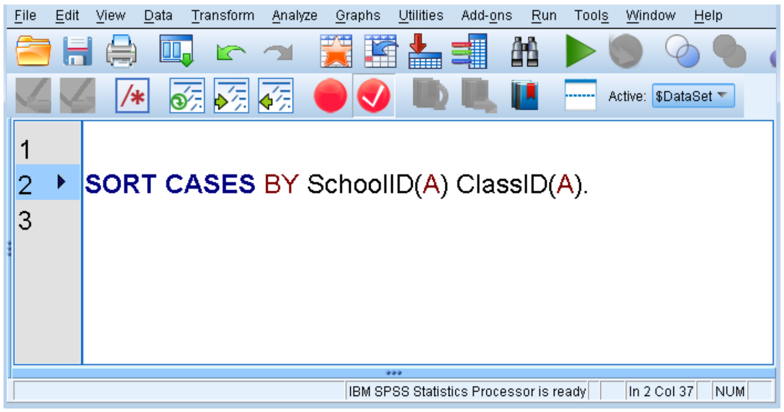
```
compute logrt002 = ln(rt002).
```

It keeps going until it goes through all of the variables in the lists. This allows the syntax to appropriately transform all 200 reaction time variables.

As mentioned before, the biggest advantage of using the **do repeat** and **end repeat** statements is that they cut down how much code you need to type. However, using them has other benefits as well. First off, it reduces the opportunities for typing errors since you are only typing the transformation in once instead of hundreds of times. Second, it makes it easier to change your program later on. If you decided to use a square root transformation instead of a logarithm, it would be easy to change this program because you'd just have to adjust line 3. If you had written out all 200 compute statements manually, you'd have to change them all individually. Finally, it makes it much easier for others to read your program and see what you're trying to do. The 200 different compute statements are really just trying to do the same thing – log transform your variables to get rid of the skewness. Paralleling this, the current program has a single statement that does all of the transformations. If you had them all typed in separately, someone else would have to read through all of them and then notice that they were doing basically the same thing, and then look for the similarities to figure out what the purpose really was.

sort cases

The **sort cases** statement is used to order the cases in your data set based on the values of particular variables. It can sort by multiple variables, such that the cases are first sorted by the values of one variable, and then any cases that have the same value on the first variable are then sorted by the second variable, and so forth. Below is an example of syntax use the **sort cases** statement.

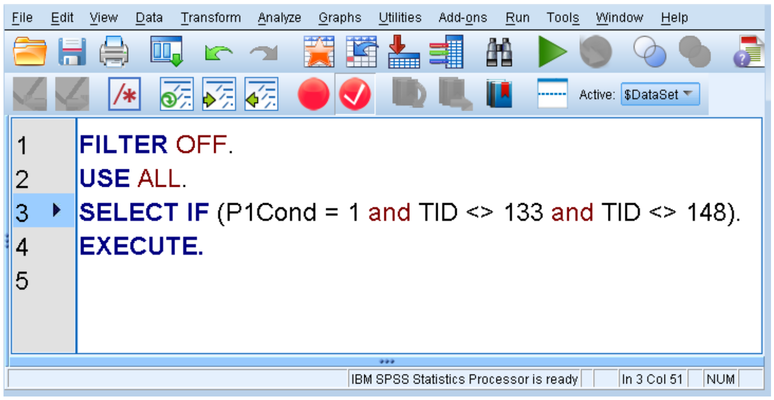


```
SORT CASES BY SchoolID(A) ClassID(A).
```

In this example, the data would first be sorted by **SchoolID**, and then within a school, the data would be sorted by **ClassID**. The **sort cases** syntax command does the same thing as the Interactive Mode procedure **Data → Sort Cases**, which was described on page 20.

select if

The **select if** statement is used to permanently remove cases from the data set based on the values of a variable in the data set. Below is an example of a program that uses the **select if** statement.


File Edit View Data Transform Analyze Graphs Utilities Add-ons Run Tools Window Help
Active: $DataSet
IBM SPSS Statistics Processor is ready
In 3 Col 51
NUM

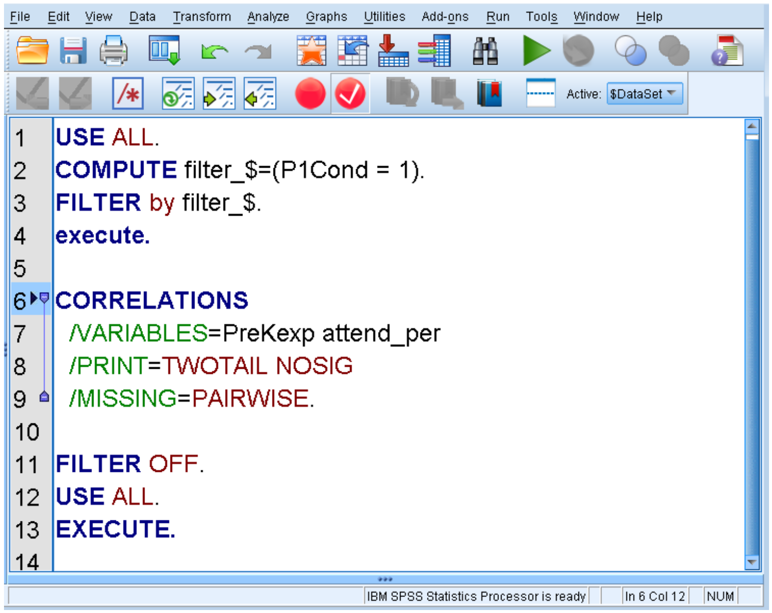
```
FILTER OFF.
USE ALL.
SELECT IF (P1Cond = 1 and TID <> 133 and TID <> 148).
EXECUTE.
```

The first two lines of the syntax remove any filtering that might've been set up prior to this command. It is a good idea to issue these two statements whenever you are about to issue a **select if** statement to make sure that you are working with the full data set. Once you run this syntax, all cases not meeting the condition in the **select if** statement will be permanently removed from the active data set, although they will not be removed from the data set on disk unless you save the dataset using the same name as the original file.

The **select cases** syntax command somewhat duplicates the Interactive Mode procedure **Data → Select Cases**, which was described on page 21, but it is only used to permanently remove cases from the data set.

filter

The **filter** statement is similar to the **select if** statement, but the **filter** statement will only temporarily exclude cases from an analysis. It is not used often in data preparation, but it is commonly used when performing analyses. Below is an example of a program that uses the **filter** statement.

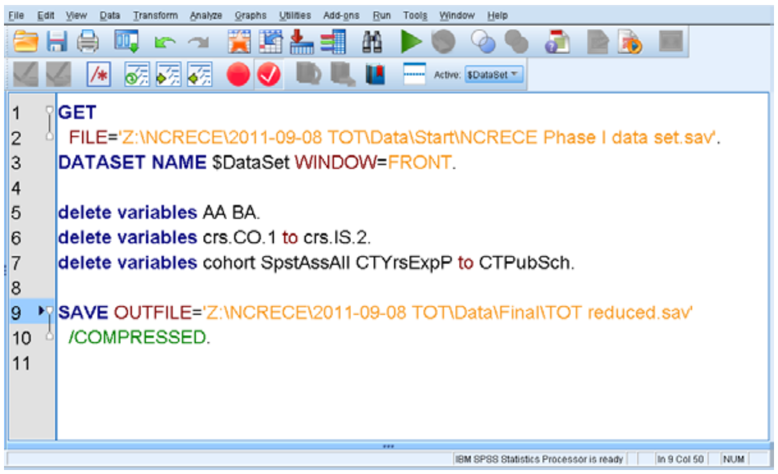

File Edit View Data Transform Analyze Graphs Utilities Add-ons Run Tools Window Help
Active: $DataSet
IBM SPSS Statistics Processor is ready
In 6 Col 12
NUM

```
USE ALL.
COMPUTE filter_$=(P1Cond = 1).
FILTER by filter_$.
execute.

CORRELATIONS
  /VARIABLES=PreKexp attend_per
  /PRINT=TWOTAIL NOSIG
  /MISSING=PAIRWISE.

FILTER OFF.
USE ALL.
EXECUTE.
```

The **compute** statement in line 2 is necessary because the **filter** statement in line 3 requires that you use a variable to define which cases are included and which cases are not. When creating your own syntax, all you need to do is replace `P1Cond = 1` with whatever criterion you want to use to determine which cases are included in analyses following the filter. After you are finished with the analyses where you want to use the filter (which would be the **correlations** command in the example), you would want to issue the statements in lines 11 through 13 to turn off the filter and use all of the cases in the data set again.

delete variables

The **delete variables** statement is used to remove variables from the data set. It can be used to remove either a single variable or a list of variables. Once you run this syntax, the listed variables will be permanently removed from the active data set, although they will not be removed from the data set on disk unless you save the dataset using the same name as the original file. Below is an example of a program that uses the **delete variables** statement.

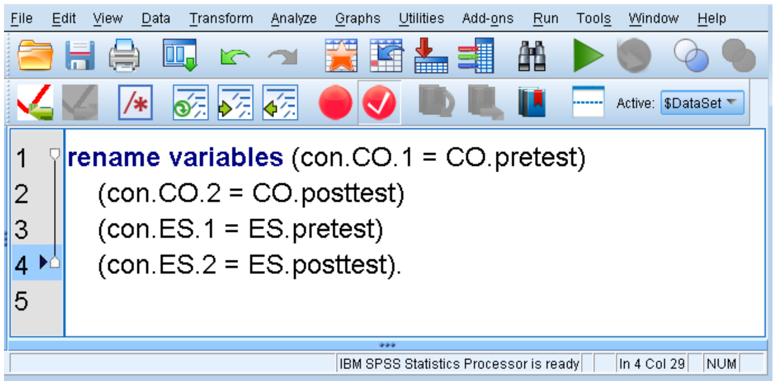

File Edit View Data Transform Analyze Graphs Utilities Add-ons Run Tools Window Help
Active: $DataSet
IBM SPSS Statistics Processor is ready
In 9 Col 50
NUM

```
1   GET
2     FILE='Z:\NCRECE\2011-09-08 TOT\Data\Start\NCRECE Phase I data set.sav'.
3   DATASET NAME $DataSet WINDOW=FRONT.
4
5   delete variables AA BA.
6   delete variables crs.CO.1 to crs.IS.2.
7   delete variables cohort SpstAssAll CTYrsExpP to CTPubSch.
8
9   SAVE OUTFILE='Z:\NCRECE\2011-09-08 TOT\Data\Final\TOT reduced.sav'
10    /COMPRESSED.
11
```

The above program shows three different ways of using the **delete variables** statement. Line 5 shows how to list out the individual variables you want to delete. Line 6 shows how to use this command to remove a set of variables that are adjacent to each other in the data set. Line 7 shows how to combine both of these in a single statement. Notice that the **save** statement starting on line 9 saves the reduced data set to a new file so that it doesn't permanently remove the information stored in the deleted variable – this way you can work with a reduced data set without losing the data stored in the unneeded variables.

rename variables

The **rename variables** statement is used to alter the names of variables in the dataset. Below is an example of a program that uses the **rename variables** statement.


File Edit View Data Transform Analyze Graphs Utilities Add-ons Run Tools Window Help
Active: $DataSet

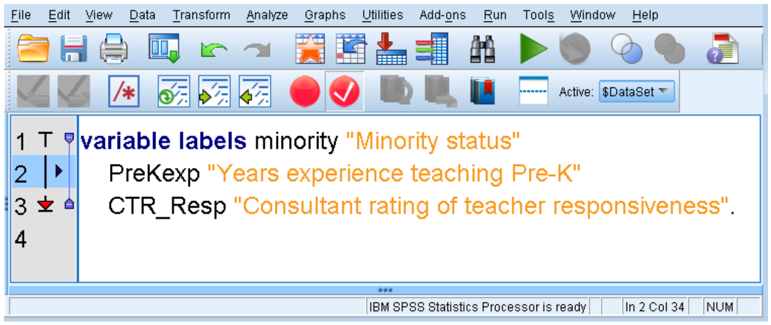
```
rename variables (con.CO.1 = CO.pretest)
   (con.CO.2 = CO.posttest)
   (con.ES.1 = ES.pretest)
   (con.ES.2 = ES.posttest).
```

IBM SPSS Statistics Processor is ready
In 4 Col 29
NUM

When using the **rename variables** command, you list out pairs of variables individually inside of parentheses. The original name of the variable is on the left of the equals sign, and the new name of the variable is on the right of the equals sign.

variable labels

The **variable labels** statement is used to assign a descriptive label to a variable. This label will be displayed in the variable view of the data set, and will also typically be used to reference the variable in the output of any analyses of the variable. It is a good idea to give descriptive labels to your variables, especially if the variable names are hard to understand. Below is an example of a program that uses the **variable labels** statement.

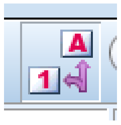
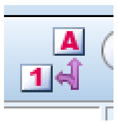

File Edit View Data Transform Analyze Graphs Utilities Add-ons Run Tools Window Help
Active: $DataSet

```
variable labels minority "Minority status"
   PreKexp "Years experience teaching Pre-K"
   CTR_Resp "Consultant rating of teacher responsiveness".
```

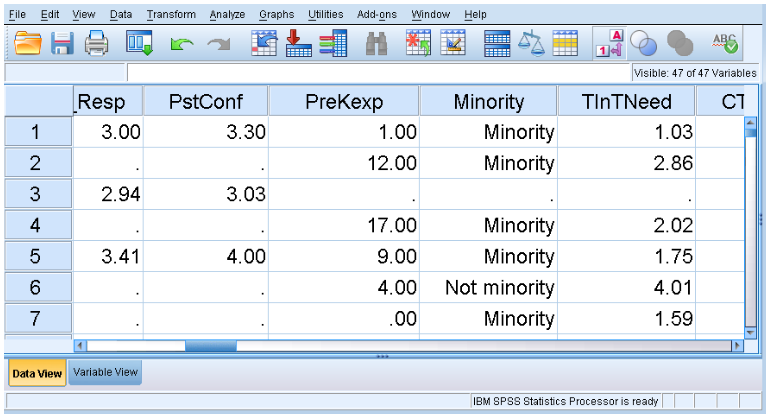
IBM SPSS Statistics Processor is ready
In 2 Col 34
NUM

value labels

The **value labels** statement is used to assign a descriptive label to the specific values that a variable can take. It is different from the variable label, which would apply to the variable as a whole. As an example, you might use a variable label to identify the variable minority as referring to a teacher's minority status, but you would use value labels to identify that a value of 1 means that the teacher is a minority and that a value of 0 means that the teacher is not a minority. Value labels will be displayed in the data view instead of the actual values if you have value labels turned on. The value labels button typically appears on the SPSS button bar. In the image below, picture on the left shows the value labels button when the labels are turned on, and the picture on the right shows the value labels button when the labels are turned off. You can click the button if you want to switch between the two modes.

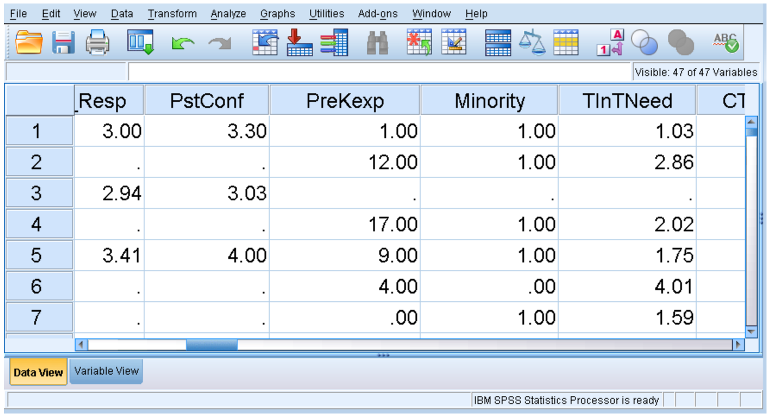



In the examples below, we added value labels to the **minority** variable. Below is an example of what the data view would look like with the value labels on.


File Edit View Data Transform Analyze Graphs Utilities Add-ons Window Help
Visible: 47 of 47 Variables
Data View
Variable View
IBM SPSS Statistics Processor is ready

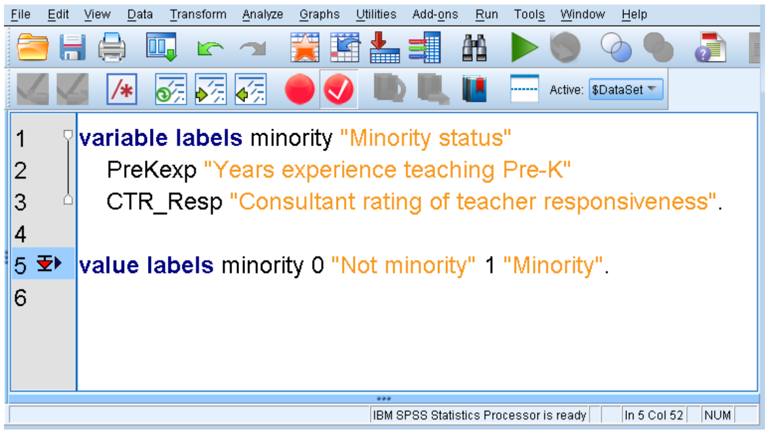
	Resp	PstConf	PreKexp	Minority	TInTNeed	CT
1	3.00	3.30	1.00	Minority	1.03	
2	.	.	12.00	Minority	2.86	
3	2.94	3.03	.	.	.	
4	.	.	17.00	Minority	2.02	
5	3.41	4.00	9.00	Minority	1.75	
6	.	.	4.00	Not minority	4.01	
7	.	.	.00	Minority	1.59	

Below is an example of what the data view would look like with the value labels off.


File Edit View Data Transform Analyze Graphs Utilities Add-ons Window Help
Visible: 47 of 47 Variables
Data View
Variable View
IBM SPSS Statistics Processor is ready

	Resp	PstConf	PreKexp	Minority	TInTNeed	CT
1	3.00	3.30	1.00	1.00	1.03	
2	.	.	12.00	1.00	2.86	
3	2.94	3.03	.	.	.	
4	.	.	17.00	1.00	2.02	
5	3.41	4.00	9.00	1.00	1.75	
6	.	.	4.00	.00	4.01	
7	.	.	.00	1.00	1.59	

Below is an example of a program that uses the **value labels** statement.


File Edit View Data Transform Analyze Graphs Utilities Add-ons Run Tools Window Help
Active: $DataSet
IBM SPSS Statistics Processor is ready
In 5 Col 52
NUM

```
variable labels minority "Minority status"
   PreKexp "Years experience teaching Pre-K"
   CTR_Resp "Consultant rating of teacher responsiveness".

value labels minority 0 "Not minority" 1 "Minority".
```