

Manipulating Variables in R

At some point after you have begun working with R, it is likely that you will want to manipulate your variables and do such things as combine vectors or group participants differently than your original grouping. If you have just started using R, this section may not seem pertinent and may be more confusing than helpful. I recommend coming back to this section as needed when R is more familiar to you. I will also note that for me personally, manipulating variables is the hardest part of using R. I will often go to a different program like Excel to do this part, because the ability to paste and copy numbers is a lot more intuitive than trying to figure out the syntax for manipulating numbers in R. Nevertheless, I want my readers to know how to do these things in R if they like, so I provide the information here in this part.

Moving or Deleting Columns or Rows

Sometimes you may want to make a large dataset more manageable with just the variables you absolutely need in it. For example, in the case of the Obarow (2004) dataset (which is discussed in Chapter 10 on Factorial ANOVA), there are 35 variables, and let's say you're really only interested in doing a statistical analysis on a few of the variables. To follow along with this analysis, import the SPSS file Obarow.Original.sav and name it **obarrow**; you should have a dataset with 81 rows and 35 columns).

To cut out an entire column of data, make sure your data is the active dataset and then use R Commander's menu sequence `DATA > MANAGE VARIABLES IN ACTIVE DATASET > DELETE VARIABLES FROM DATASET`. Let's say that we want to cut out the variable grade.

All we do is choose it out of the list of variables. A prompt comes up to verify that I want to delete it, and I press OK. If you want to delete more than one variable, just hold the Ctrl button down and click on additional variables.

The R code for this action is quite simple:

```
obarrow$grade <- NULL
```

You just assign nothingness to the variable, and it ceases to exist.

Sometimes you may have a principled reason for excluding some part of the dataset you have gathered. If you are getting rid of outliers, robust statistics (which I will discuss throughout this book) are probably the best way to deal with such a situation. However, there are other situations where you just need to exclude some part of the dataset *a priori*. In the Obarow (2004) study, some children who participated in the experiment on vocabulary learning achieved very high scores on the vocabulary pretest. Children with such high scores would not be able to achieve many gains on a posttest, and one might then have a principled reason for cutting them out of the analysis (although you should tell your readers that you did this). This would mean cutting out certain rows in the dataset.

If you already know in advance which rows you want to remove, the process is straightforward. Follow the menu command in R Commander: DATA > ACTIVE DATASET

> REMOVE ROW(S) FROM ACTIVE DATASET. Looking at the window labeled “Indices or quoted names of row(s) to remove” in Figure 1 you could specify only one row to remove (put in a single number, such as “7”, to remove row 7), a range of rows (use a colon to specify the range, such as rows 7 to 11 by writing “7:11”), or a number of different rows (here use the concatenation function by adding a “c” before the number of the rows, all listed in the parentheses, like this “c(3,6,7,11)”). You also have the option to save this file under a different name.

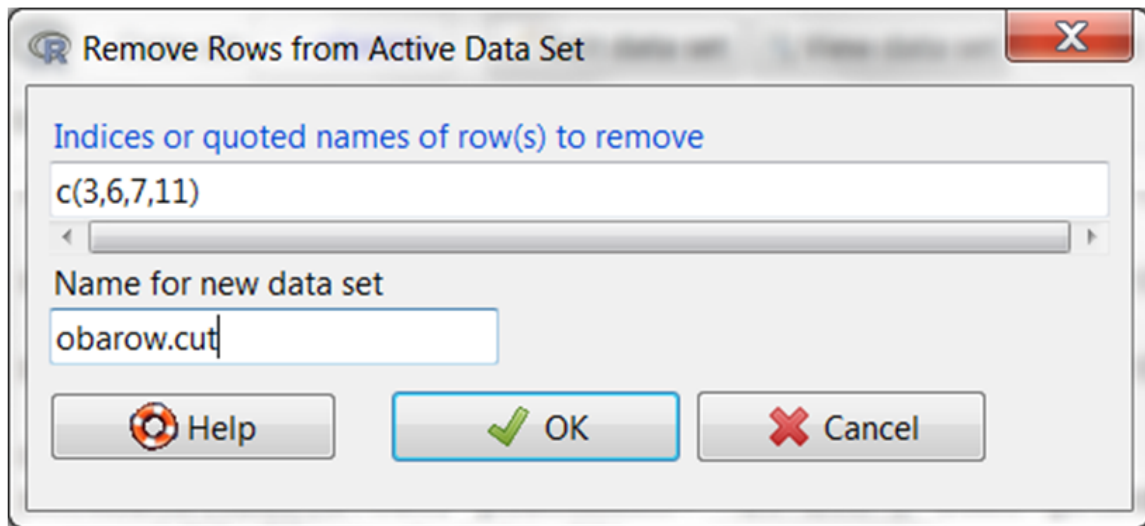


Figure 1 Removing a row from the active dataset in R Commander.

The R code for performing this action is simple as well, as long as you understand that in R you specify rows and columns in brackets, with rows first and columns second. The R code for removing rows 7 through 11 from the **obarow** dataset and creating a new one is:

```
NewObarow<-obarrow[-c(7:11),]
```



Tip: Here's a silly mnemonics device to help you remember which comes first, the row or the column:
Row, row, row your boat gently down the stream, toss your column overboard and listen to it scream.
Row comes first and column comes second in my version of the nursery rhyme, so you can remember it works that way in the syntax too.

R uses the minus sign (“-”) to show that it will subtract this subset of rows. The concatenation function, `c()`, is used whether or not there is more than one row or column being deleted. So the R command says to subtract out rows 7 through 11 (we know they are rows because they come before the comma) from all of the columns (we know this because nothing is specified after the comma; if you wanted to subtract out certain rows only from certain columns you would specify the columns after the comma).

But what if you don't know exactly which rows you want to delete? What if, instead, you want to remove any data that exceeds a certain number? To do this, you can use the R console and the `subset()` command. The following command puts a subset of the original dataset into a file with a different name. In this command I get rid of any cases where the `Pretest 1` score was over 18 points (out of 20 possible).

```
NewObarow<-subset(obarrow, subset=pretest1<=18)
```

Notice that in the subset command I have used the comparison operators less than or equal to (" \leq "). Other such operators in R include more than (" $>$ "), more than or equal to (" \geq "), less than (" $<$ "), equal to (" $==$ "), or not equal to (" \neq "). In this case, any row where the pretest1 score is over 18 will be cut from the dataset.

Summary: Deleting Parts of a Dataset

In R Commander, for removing columns, choose DATA > MANAGE VARIABLES IN ACTIVE DATASET > DELETE VARIABLES FROM DATASET

For removing rows choose:

DATA > ACTIVE DATASET > REMOVE ROW(S) FROM ACTIVE DATASET

To remove one number, write in the number ("7")

To remove a range, put a colon between two numbers (7:11)

To remove selected rows, use the `c()` command ("`c(3,6,7,11)`")

In R, to remove columns, assign the column to the NULL operator (N.B. items in red should be replaced with your own data names):

```
obarrow$grade <- NULL
```

To remove rows specify which rows to subtract out (and possibly rename your data):

```
NewObarow<-obarrow[-c(7:11), ]
```

Deleting Parts of a Dataset Practice Activities

- 1 **Working with rows and columns.** First, let's do some of the examples with data that are found in R itself. These examples use a dataset that comes built-in in R called `swiss`, which has data on Swiss fertility and socioeconomic indicators for various cities (use the command `library(help="datasets")` to see a list of all datasets that are included in R). This is a rather large dataframe, so let's first cut it down to make it more manageable. Type:

```
sw<-swiss[1:5, 1:4] #this cuts down to the first 5 rows and the first 4 columns
```

```
sw #look at your data now
```

Now we will work with the dataset **sw**.

Perform the following commands, and explain what happens for each one,

following the example answer given in (a):

- a. `sw[1:3]` #shows first 3 columns, all rows
 - b. `sw[1:3,]` #
 - c. `sw[,1:3]`
 - d. `sw[4:5, 1:3]`
 - e. `sw[[1]]`
 - f. `sw[[5]]`
 - g. `sw["C",]`
- 2 In exercise #1 you created a new data frame called `sw`, with 4 variables. Remove the variable **Examination**.
- 3 Use the dataset **ChickWeight** (this is a built-in R dataset). This data looks at the weight versus age of chicks on different diets. How many data points are there? Find out using the `length()` command with any variable in the dataset (don't just use the name of the dataset itself, or it will only be 4!). Create a new variable called **ChickDiet** that excludes chicks who received Diet #4. How many participants are in the **ChickDiet** dataset?

Combining or Recalculating Variables

For this example we will use a dataset from Torres (2004). Torres surveyed ESL learners on their preference for native speaking teachers in various areas of language teaching.

This file contains data from 34 questions about perception of native- versus non-native-

speaking teachers. Let's say that I am interested in combining data from 5 separate questions about whether native-speaking teachers are preferable for various areas of language study into one overall measure of student preference. I want to combine the 5 variables of Pronunciation, Grammar, Writing, Reading and Culture, but then average the score so it will use the same 1–5 scale as the other questions.

To do this in R Commander, first make sure the dataset you want to work with is currently active and shown in the “Dataset” box in R Commander (you will need to import it from the SPSS file Torres.sav; call it **torres**). Combine variables by using DATA > MANAGE VARIABLES IN ACTIVE DATASET > COMPUTE NEW VARIABLE. You will see a computation box like the one in Figure 2 come up:

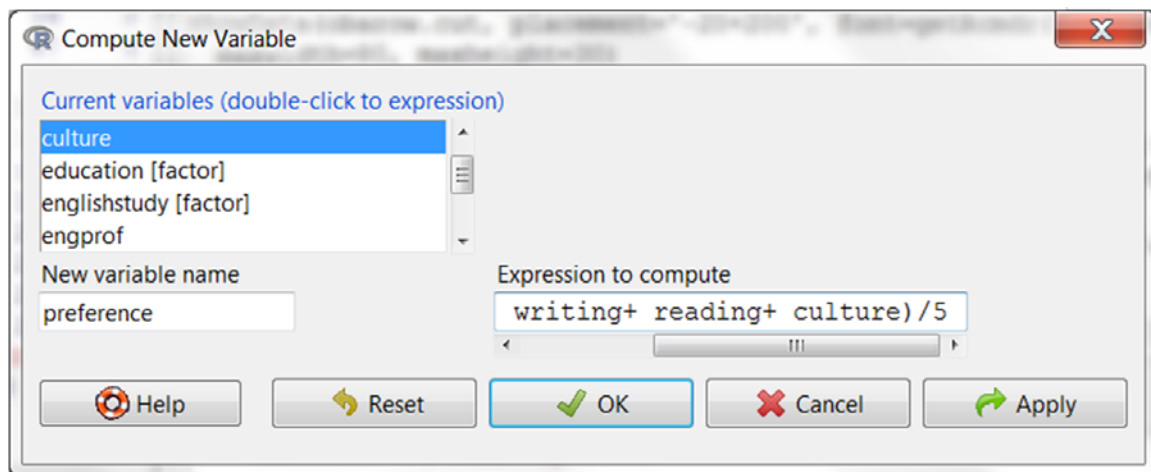
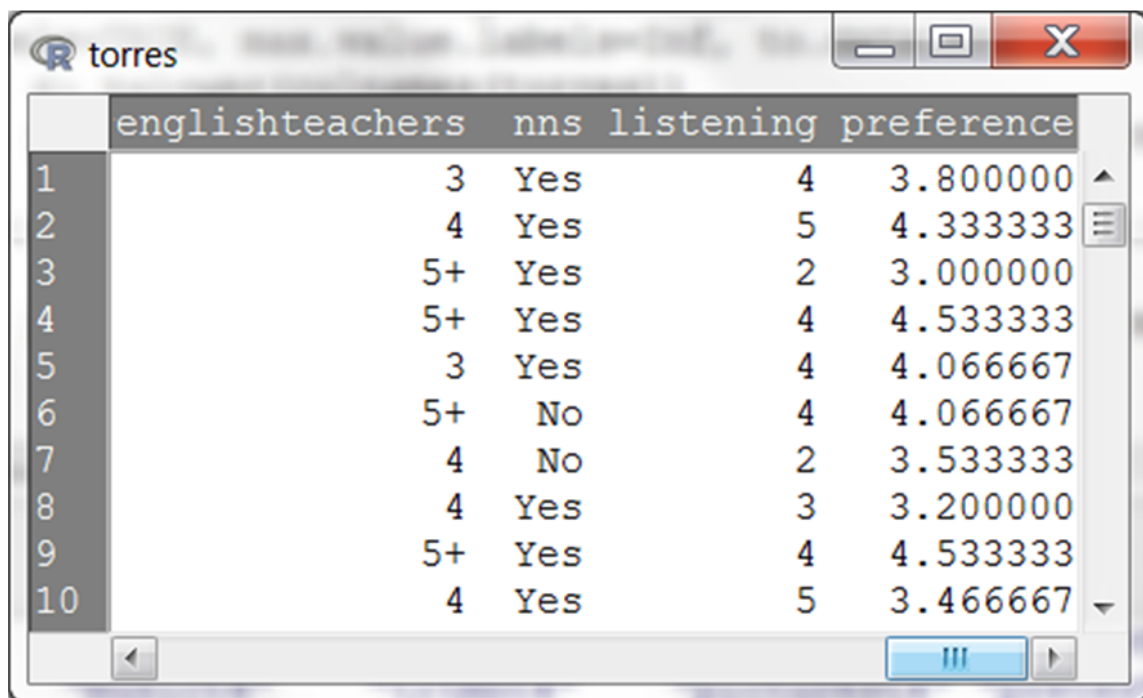


Figure 2 Computing a new variable in R Commander.

As you can see in Figure 2, I called the new variable **preference**. I moved the 5 variables that make up this new variable from the “Current variables” box into the “Expression to compute” box by double-clicking on them. After starting out with a parenthesis at the beginning, I added the variables one by one. After each variable I manually typed in a

plus sign, and at the end of the entire string I typed in “)/5” to divide it all by 5. You can use the same symbols (such as “+”, “*”, “^”) that were explained in Section 1.5.1 of the book to perform mathematical computations. Last of all you will want to make sure this worked right by opening up the “View dataset” button on the R Commander interface and scrolling over to the end of the dataset. The new column gets appended to the end of the dataset. Figure 3 shows how I have done this, and the column appears to be appropriately measured on a 5-point scale.



	englishteachers	nns	listening	preference
1	3	Yes	4	3.800000
2	4	Yes	5	4.333333
3	5+	Yes	2	3.000000
4	5+	Yes	4	4.533333
5	3	Yes	4	4.066667
6	5+	No	4	4.066667
7	4	No	2	3.533333
8	4	Yes	3	3.200000
9	5+	Yes	4	4.533333
10	4	Yes	5	3.466667

Figure 3 Verifying that a new variable was correctly calculated in R Commander.

I also ask for the range of scores to make sure I'm on the right track:

```
range(torres$preference)
```

```
[1] 2.266667 5.000000
```


Looks good! I will also show you how commands can be executed on the R Console. As we go along I will first show you how to perform an action using R Commander, but then I will also explain the R code as well. At first you may not care about this, but as you become more experienced with R you might want to begin to move toward using more code, and so I want you to understand it. Note that the R code can be found in R Commander in the Script window. If you want to take the code that R Commander used for a certain command and tweak it a little, you can simply copy the code from the Script window to the console and paste it in.

Every time I give you R code, I will try to explain it in more detail. The box below first gives the command that was used to combine the variables. I then try to break down the command into its constituent parts and explain what each part is doing.

```
torres$preference <- with(torres, (pron+ grammar+ writing+ reading+ culture)/5)
```

```
torres$preference
```

This command creates and names the new variable
preference

```
<-
```

This symbol says to assign everything to the right of it into the expression at the left; you can also use an ‘equals’ sign (“=”)

```
with(torres, (expression) )
```

```
with(data, expression, . . .)
```

The **with()** command says to evaluate an R expression in an environment constructed from data

```
(pron+grammar+ writing+  
reading+ culture)/5
```

This is the arithmetic expression

Summary: To use existing variables to calculate a new variable

In R Commander, choose: DATA > MANAGE VARIABLES IN ACTIVE DATASET >

COMPUTE NEW VARIABLE

Use ordinary arithmetic expressions to combine variables or make calculations

In R, use the template:

```
torres$preference <- with(torres, (pron+ grammar+ writing+ reading+  
culture)/5)
```

Combining or Recalculating Variables Practice Activities

- 1 Use the **torres** dataset. From this dataset, create a new variable called **OralPreferences** by combining the variables of speaking and listening preference for a native-speaker teacher. Don't forget to average the two scores so the result stays on a 1–5 scale. What is the range of scores for this new variable? Find out by using the function **range()** on the new variable.
- 2 Use the dataset **mcguireSR** (this is a text file, comma delimited, if you did not import it earlier in the chapter). Create a variable called **gainscore** by subtracting the pretest from the posttest scores. What is the mean score for this variable?
- 3 Use the dataset **partial** (this is an SPSS file, called LarsonHall.partial.sav; import it now, if you have not imported it before). The variable of **r_l_accuracy** is measured in its raw form as a score out of 7. Convert the scores to percentages out of 100 and call the new variable **PercentAccuracy**. What is the maximum percentage achieved?

Recoding Group Boundaries

Sometimes you want to take the data and make categorical groups from it (you should be careful though; making categorical groups when you have interval-level data is just throwing away information). To illustrate this process, let's look at data from DeKeyser (2000). DeKeyser administered a grammaticality judgment test to a variable of child and adult Hungarian L1 learners of English. DeKeyser divided the participants on the basis of whether they immigrated to the US before age 15 or after (this is his **Status** variable). But let's suppose we have a good theoretical reason to suspect that there should be 4 different groups, and we want to code the data accordingly (I want to make it clear that I don't actually think this is a good idea for this particular dataset; I'm just using it as an example).

In R Commander, we can create new groups (also called recoding variables) by first making sure the dataset we want is the active one in the "Dataset" box in R Commander (if you have not already done so for a previous application activity, in order to follow along with me here you'll need to import the DeKeyser2000.sav SPSS file; name it **dekeyser**). Then pull down the DATA menu and choose MANAGE VARIABLES IN ACTIVE DATASET > RECODE VARIABLES. A dialogue box will open (see Figure 4).

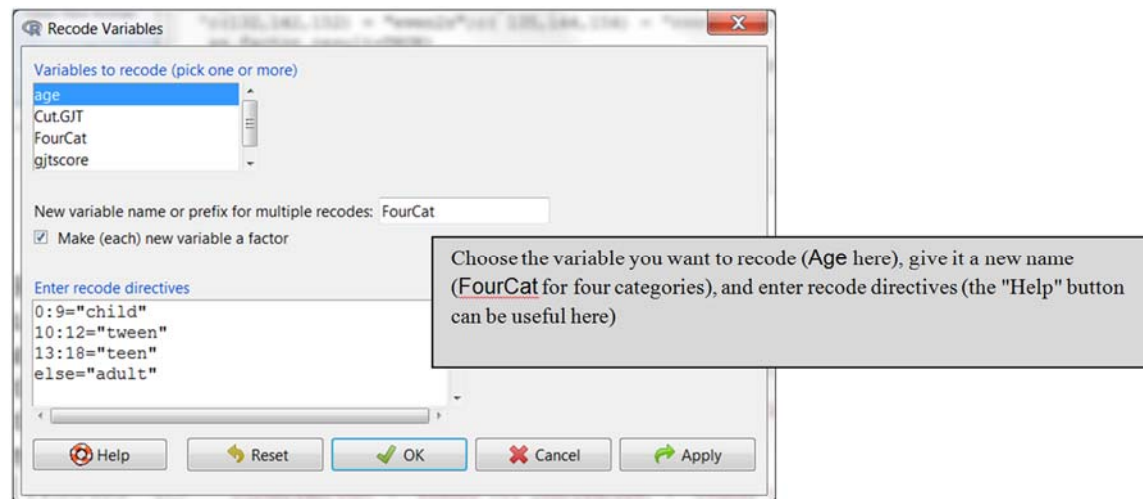


Figure 4 Recoding variables in R Commander.

The recode directive is the most important part of this process. Make sure to put the name of the new category in between parentheses. You can choose individual numbers separated by commas (1,2,3) or ranges (as shown in Figure 4) as ways of specifying data. The phrase “else” can be used to recode everything that has not already been specified. After I pressed OK I then looked at the dataset and verified that this new variable had been created and seemed fine.

The R code for this command is:

```
dekeyser$FourCat <- recode(dekeyser$Age,  
'0:9="child"; 10:12="tween"; 13:18="teen"; else="adult"; ',  
as.factor.result=TRUE)
```

dekeyser\$FourCat	This command creates and names the new variable FourCat
<-	Assignment operator
recode(data, expression, as.factor.result=TRUE)	The recode command gives instructions on how to restructure a numeric vector into one that is a factor. The expression part gives the recode specifications
'0:9="child"; 10:12="tween"; 13:18="teen"; else="adult"; '	This is the recode specification; note that the entire expression is enclosed in single quotes; each part of the recode directive is separated by a semi-colon; the expression 0:9="child" specifies that if the number is 0 through 9, the label child (and so on) should be attached in this factor; the expression else="adult" means I don't have to specify how high the ages go after age 18

If you want to change existing factor levels that are already categorical and named with non-numerical characters, you will need to put quotation marks around the existing names as well as the new names, as shown in the recoding done for the `beq` file (shown in Figure 5).

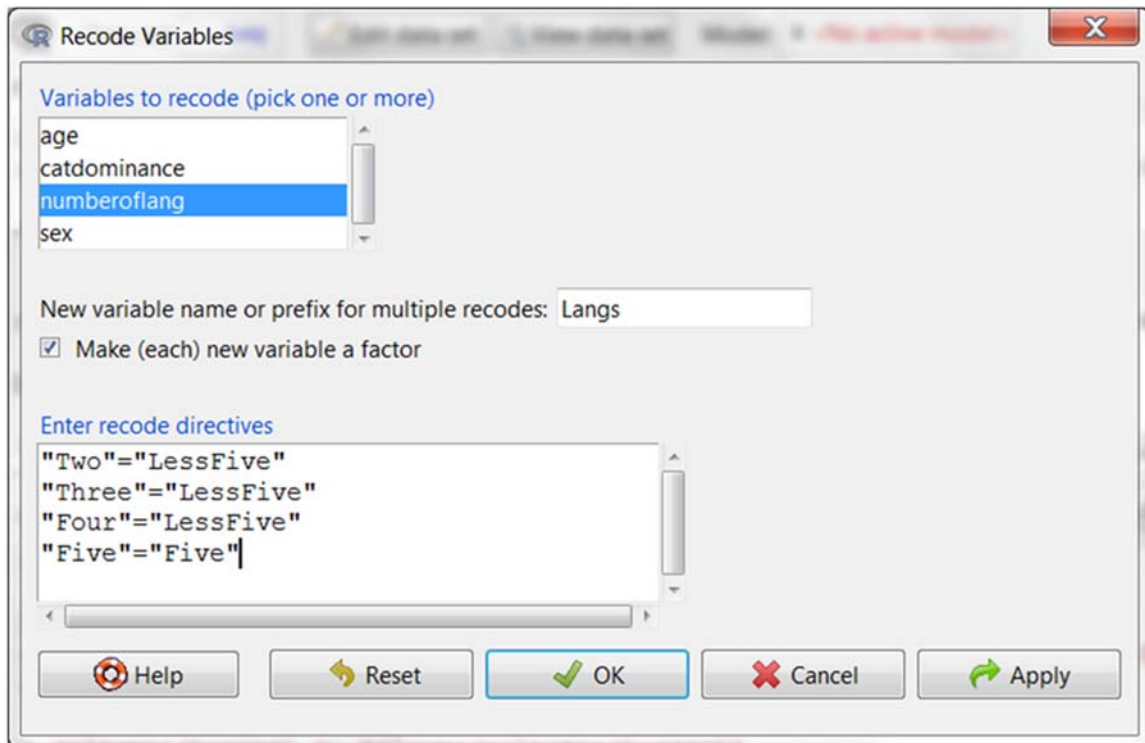


Figure 5 Recoding character variables in R Commander (factors with labels).

The R code for this command is:

```
beq$Langs <- recode(beq$NumberOfLang,
  "Two"="LessFive"; "Three"="LessFive"; "Four"="LessFive"; "Five"="Five"; ,
  as.factor.result=TRUE)
levels(beq$Langs) #Use this to check that everything came out right
[1] "Five" "LessFive" #Looks good!
```

Summary: Creating categorical groups from existing data

In R Commander, choose:

DATA > MANAGE VARIABLES IN ACTIVE DATASET > RECODE VARIABLES

In recode directives, enter a range of numbers to the left of the equal sign (separate range with colon) and enter the name of the new category in quotes to the right of the equal sign. If you are recoding categories that already exist, put parentheses around character symbols to the left of the equal sign as well.

In R, use the template:

```
beq$Langs <- recode(beq$NumberOfLang,  
  "Two"="LessFive"; "Three"="LessFive"; "Four"="LessFive";  
  "Five"="Five"; ',  
  as.factor.result=TRUE)
```

Recoding Group Boundaries Practice Activities

- 1 Using the beq dataset, recode the variable **NumberOfLang** so there are only two levels: Those with two or three languages (call them “minorglots”) and those with four or more languages (call them “majorglots”). Call your new variable **Glots**. Use the **summary()** command to see how many participants are in each of your new categories. Note: You might want to reimport this file (BEQ.Dominance.sav) so that all of the cases will be there, since if you were following along with me in the text you might have deleted some cases in previous activities.
- 2 Use the **mcguireSR** dataset (import as a text file; it is comma-delimited). Currently the participants are divided into whether they were in the experimental or control group. But let’s say you decided you wanted to divide them into slow and fast speakers, according to their fluency scores on the pretest (again, I don’t think this is a good idea, but it’s just for practice in manipulating variables!). Divide the speakers by calling those

lower than the mean score “slow” and those at the mean or higher “fast”.

Name your new variable **rate**. Verify your variable has two levels by typing the name of the new variable, and use the **summary()** command to see how many participants are in each of your new categories.

- 3 Use the **torres** dataset (import SPSS file Torres.sav), and the variable labeled **beginner**. This indicates whether each person prefers a NS teacher for a beginning-level language learner (where 5 = strongly prefer, 3 = neither prefer nor dislike and 1 = strongly dislike). Categorize participants into those who have strong opinions (both 5s and 1s), moderate opinions (2s and 4s) and neutral (label them as ‘strong’, ‘moderate’, and ‘neutral’). Call this new variable **TypeOfPreference**. Which type of participant predominates? Use the **summary()** command to see how many participants are in each category.

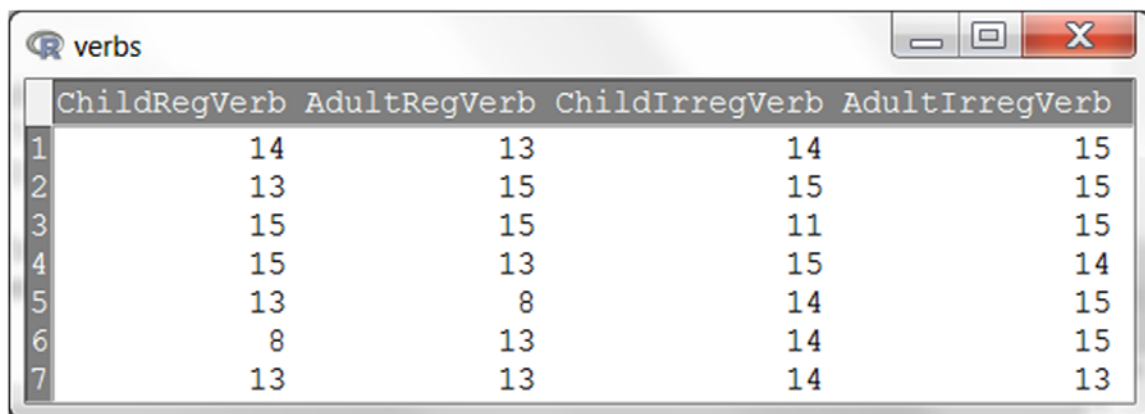
Getting Your Data in the Correct Form for Statistical Tests

There are two basic ways that you might have your dataset up:

- 1 Data is split so that the results for each group for each variable are found in different columns. We’ll call this the ‘**wide**’ form (see Figure 6). This form of data which is already split by the categorical variables is often used for the robust statistics tests in the WRS package created by Wilcox (2005, 2012) that are used in this book.
- 2 All the data for one variable is in one column, and there is another column that codes the data as to which group it belongs to. Everitt and Dunn

(2001) call this the ‘long’ form because the columns will be longer in this case. This is the form used for ANOVA analysis (see Figure 7).

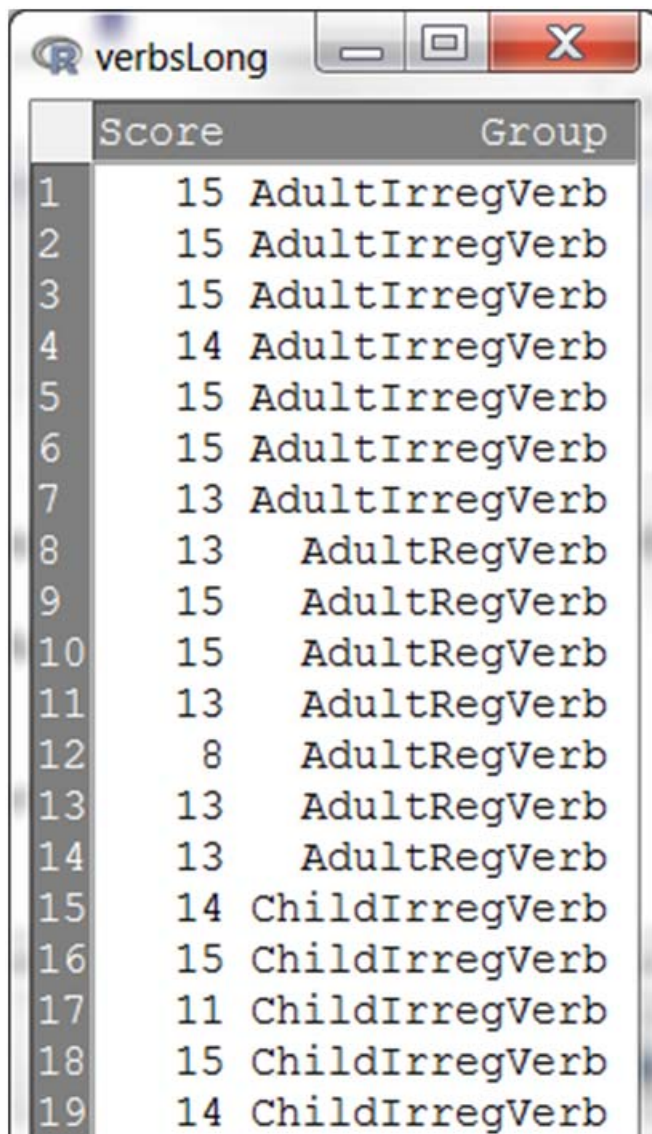
Here is an example of the data in the wide format. Let’s say we are looking at the correlation between test scores of children and adults on regular and irregular verbs. We would have one column that represented the scores of the children on regular verbs, another column containing the scores of adults on the regular verbs, etc. In the wide format, we do not need any indexing (or categorical) variables, because the groups are already split by those variables (adult vs. child, regular vs. irregular verbs) into separate columns (see Figure 6; note that I just typed this data in by hand—it does not come from any dataset. If you’d like to follow along with me, just type the initial data in Figure 6 in yourself and call the file **verbs**).



	ChildRegVerb	AdultRegVerb	ChildIrregVerb	AdultIrregVerb
1	14	13	14	15
2	13	15	15	15
3	15	15	11	15
4	15	13	15	14
5	13	8	14	15
6	8	13	14	15
7	13	13	14	13

Figure 6 Data in the ‘wide’ format.

This data can be put into the long format (see Figure 7). In this case all of the scores are put into just one column, and there is another column (which is a factor) that indexes both the group variable and the verb regularity variable at the same time.



	Score	Group
1	15	AdultIrregVerb
2	15	AdultIrregVerb
3	15	AdultIrregVerb
4	14	AdultIrregVerb
5	15	AdultIrregVerb
6	15	AdultIrregVerb
7	13	AdultIrregVerb
8	13	AdultRegVerb
9	15	AdultRegVerb
10	15	AdultRegVerb
11	13	AdultRegVerb
12	8	AdultRegVerb
13	13	AdultRegVerb
14	13	AdultRegVerb
15	14	ChildIrregVerb
16	15	ChildIrregVerb
17	11	ChildIrregVerb
18	15	ChildIrregVerb
19	14	ChildIrregVerb

Figure 7 Data in the 'long' format.

With the help of R Commander, moving from one form to another is not too difficult.

This section will explain how to do this with R Commander's `stack()` command, but

also see Section 11.2.3, which uses a different command called `melt()` to change data from the wide form to the long form.

To go from the wide form to the long form, in R Commander I went to DATA > ACTIVE DATASET > STACK VARIABLES IN ACTIVE DATASET. There I picked all 4 variables in the `verbs` dataset by holding down the CTRL button on my keyboard and right-clicking my mouse on each variable. I renamed the entire file `verbsLong`, renamed the numeric variable `Score` and the factor variable that would be created `Group`. Figure 8 shows the “Stack Variables” dialogue box, and Figure 7 is the result from that command.

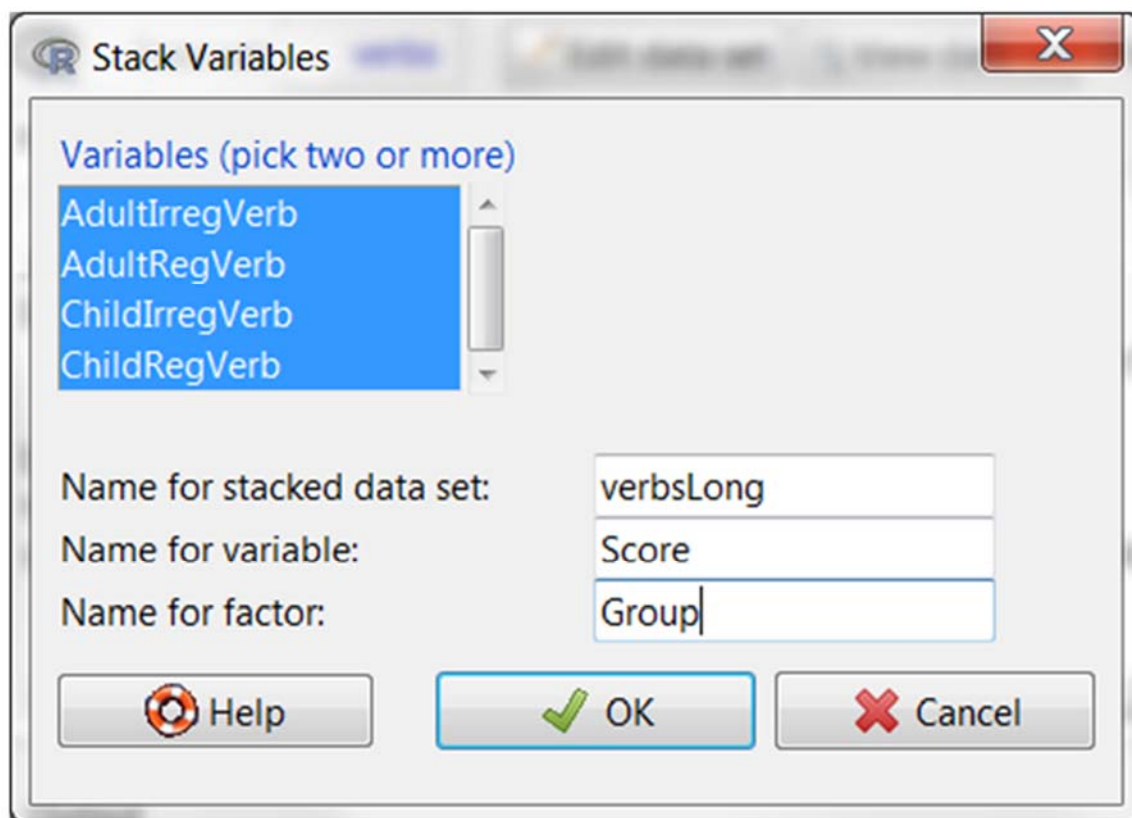


Figure 8 Going from wide form to long form in R Commander.

This process can be done using R code:

```
verbsLong <- stack(verbs[,
```

```
c("AdultIrregV", "AdultRegVerb", "ChildIrregVerb", "ChildRegVerb"))
```

```
names(verbsLong) <- c("Score", "Group")
```

```
verbsLong<-
```

Assign the result of what's on the right-hand side of the assignment operator to the object `verbsLong`.

```
stack(verbs[ ])
```

This command stacks separate vectors on top of each other and then creates an indexing column; here, it is told to work on the dataset `verbs`.

```
[, c("AdultIrregV", "AdultRegVerb",  
"ChildIrregVerb", "ChildRegVerb")]
```

The data inside the brackets specifies that we want all of the rows from the dataset `verbs`, and the 4 columns that correspond to the ones listed.

```
names(verbsLong )
```

The names on the right-hand side ("`Score`", "`Group`") are assigned to the names dimension of the newly created `verbsLong` data frame.

For moving from a long format to a wide one, you can simply subset the original dataset along the categorical variable or variables. In R Commander, choose DATA > ACTIVE DATASET > SUBSET ACTIVE DATASET. The tricky part comes in figuring out what to put in

the box “Subset expression.” This needs to be set up so that the level of the categorical variable is specified. Therefore, before you open the dialogue box to do this, makes sure to find out what the levels are for your variable.

```
levels(verbsLong$Group)
```

```
[1] "AdultRegVerb" "ChildRegVerb" "AdultIrregV" "ChildIrregVerb"
```

First, I click off “Include all variables” and select the variable of “Score” because I only want *one* column of data, not a column with Group and a column with Score. For the subset expression, notice that I need to use double equal signs (“==”) after the name of the variable (“Group”) and that I need to spell and capitalize the name of the level exactly right, and put it in parentheses. Figure 9 shows the dialogue box and resulting dataset.

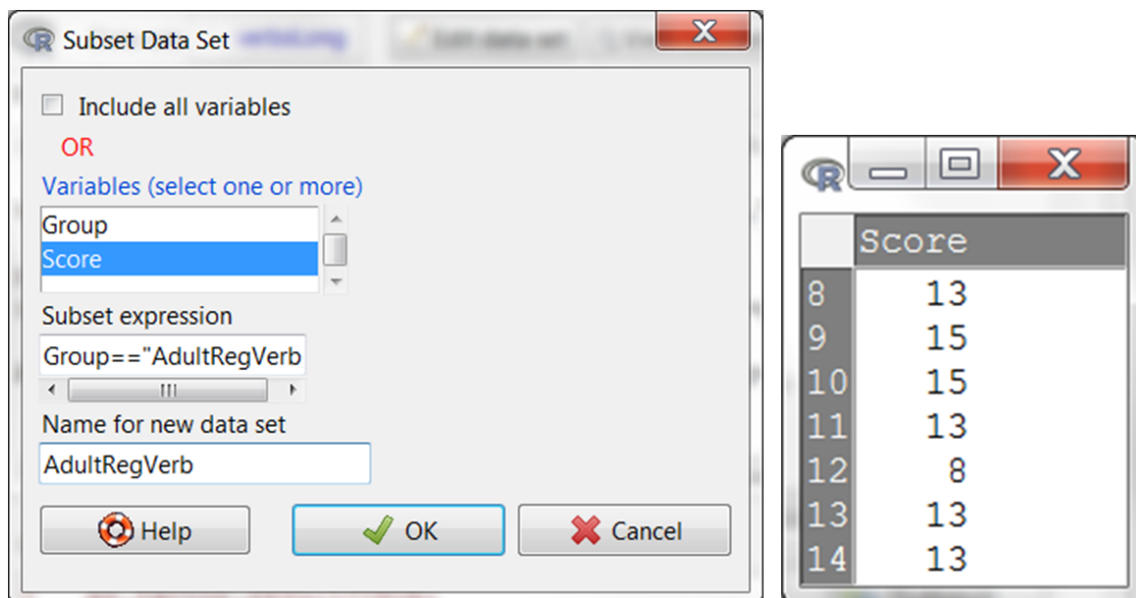


Figure 9 Subsetting a dataset in the long form.

In order to recreate the wide form, you'd then need to continue on with this process, subsetting all of the different columns that you need (you can see that for this small dataset, it would be much easier to simply type the numbers in by hand!). If you use this process, don't forget to change your active dataset in R Commander *back* to the original dataset (in this case, the one called `verbsLong`) before subsetting another column of data. If you try to subset the dataset you just created (in my case, `AdultRegVerb`) your result will be null! Click in the "Dataset" box along the top of the R Commander window to go back and choose the original dataset. In the end I will have four different datasets that I will want to put together into one wide dataset. What is needed is to coerce the data together into a data frame and correctly label the columns. There's no way I know of in R Commander to do this, but this R code will do it:

```
Verbs<-cbind.data.frame(ChildRegVerb,AdultRegVerb,ChildIrregularVerb,
AdultIrregularVerb)
```

`Verbs<-`

Assign the result of what's on the right hand side of the assignment operator to the object `Verbs`.

`cbind.data.frame()`

`cbind` stands for "column bind," so this ties together columns and makes them into a data frame

`ChildRegVerb,AdultRegVerb,`

The names of the 4 columns I want to bind together

`ChildIrregularVerb, AdultIrregularVerb`

The next step is to name all of the columns the way I want:

```
dimnames(Verbs)[[2]]=c("ChildRegVerb","AdultRegVerb","ChildIrregVerb","AdultIrregVerb")
```

<code>dimnames(Verbs)</code>	Set the dimension names of an object, in this case, the data frame <code>Verbs</code> .
<code>[[2]]</code>	Specifies that the dimension should be the columns (remember rows come first, then columns).
<code>c("ChildRegVerb","AdultRegVerb","ChildIrregVerb","AdultIrregVerb")</code>	The concatenation command <code>c()</code> contains the names I want appended to the columns. Note these must have quotes around them!

If you have followed my computations, you can verify for yourself at this time that the dataset `Verbs` looks just like it started out in Figure 5. One major problem with this process is that to make a data frame, the vectors you are binding together must be exactly the same length. If they are not, you will get an error message that tells you the vectors have differing numbers of rows. But actually in most cases, you don't need to bind the subsetting vectors together—as a single vector they will perform in the statistical command the way you intended.

To subset using R code, use the `subset()` command. Here is an example of one I used with this data.

```
ChildRegVerb <- subset(VerbsLong, subset=Group=="ChildRegVerb",  
select=c(Score))
```

ChildRegVerb<-	Assign the result of what's on the right hand side of the assignment operator to the object ChildRegVerb .
subset()	Take a subset of the first argument
VerbsLong	The name of the dataset I want to subset
subset=Group=="ChildRegVerb"	Name the level of the variable you want to subset out. Be careful to use the double equal signs and the quotes around the name of the level, otherwise you'll get an error message
select=c(Score)	Include this if you don't want to take all of the columns from the original dataset. Here I only wanted the column with the Score

In summary, manipulating data from the wide form to the long or vice versa is not impossible in R, but it is a process requiring a number of steps. Certainly if you cut and paste data this may be a much easier process in another program that works with spreadsheets such as Excel, depending on how large your dataset is.

Getting Data in the Correct form for Statistical Tests Application Activity

- 1 Open the **dekeyser** dataset (it is an SPSS file). It is currently in the long form (you could look at it by using the command **head(dekeyser)** to just see the first 6 rows). Subset it into the wide form so there are only 2 columns of data, one for participants “Under 15” and one for those “Over 15” for the **gjtscore** variable. The resulting vectors are of different lengths so do not try to combine them together. How many entries are in each of the 2 subsetted vectors (you can use the **length(DkCats1\$gjtscore)** command or the Messages section of R Commander will tell you how many rows the new subset contains)?
- 2 Import the Obarow.Story1.sav SPSS file (call it **OStory**). This file is in the long form. Subset it into 4 columns of data, each one with results on the first gainscore (**gnsc1.1**) for each of the 4 possible conditions (**+pictures**, **+music**) listed in the **Treatment** variable. How many entries are in each of the 4 subsetted vectors (you can use the **length()** command or the Messages section of R Commander will tell you how many rows the new subset contains)?
- 3 Open the .comma delimited (.csv) text file called **Effort**. This is an invented dataset comparing outcomes of the same participant in 4 different conditions. It is set up in the wide form. Change it to the long form so there are only 2 columns of data and call it **EffortLong**; call one column **Score** and the other **Condition**. What is the average score over the entire group (this is not really something you’d be interested in, but it’s just a way to check you’ve set up the file correctly)?

- 4 Open the comma delimited (.csv) text file called **Arabic**. This is an invented dataset that looks at the results of 10 participants on 4 phonemic contrasts in Arabic, and also records the gender of each participant. It is set up in the wide form. Change it to the long form so there are only 2 columns of data that are indexed by the type of contrast that was used, and call it **ArabicLong**. Call one column **Contrast** (as in the phonemic contrast) and the other **Score**. What is the average score over the entire group (this is not really something you'd be interested in, but it's just a way to check you've set up the file correctly)?

Splitting Files by Groups

Many times you will want to perform statistical operations on data split by groups. Some commands in R Commander provide a way for you to split data into groups easily by building it into the dialogue box. However, other commands do not have this built in. When you want to split data on your own you then have two options: 1) You can split your original data file into parts for each group. You will then be working with separate datasets, and will have to remember to change the active dataset each time in R Commander; or you can 2) Specify the row numbers for each group in square brackets after the name of the data file, using the command in R console. The second method is much easier I think, but requires that you move to R console, which you might not want to do.

To split the original data file into separate files, follow the R Commander menu DATA > ACTIVE DATASET > SUBSET ACTIVE DATASET. If you want to subset the entire file, keep the box called “Include all variables” checked. If you only want to make a new dataset for some of the variables, check off that box and select only the variables you want. The tricky part for this dialogue box is the “Subset expression” box. This needs to be in the form of the exact name of your splitting variable (your group variable) followed by two equals signs, and then the exact name of your group in parentheses. The section titled “Getting your data in the correct form for statistical tests” showed this process for the file `verbsLong` (see Figure 9 for the Subset Dataset dialogue box). I will give another example here. In the application activities you have worked with the DeKeyser data. Let’s say you want to split this file by the groups. First look at the names of the variables by opening up the “View dataset” button on R Commander, or typing `names(dekeyser)` into R console. Here we will want to divide `gjtscore` by `status`, so we have to look at the exact name of the groups in the status variable. Again, do this in the “View dataset” mode in R Commander, or type `levels(dekeyser$status)` into R console and see what names are given.

```
> names(dekeyser)
[1] "age"      "gjtscore" "status"
> levels(dekeyser$status)
[1] "Under 15" "Over 15"
```

If you’re using R Commander then, in the “Subset expression” box you’d have to now create this string:

```
status=="Under 15"
```

for the “Under 15” group. You should then give this dataset a new name, such as `dekeyser.under15`. I’ve shown the dialogue box for this process in Figure 10. Notice that if you forgot the space between “Under” and “15”, or didn’t capitalize the “U” in “Under”, R Commander would execute your command without any complaints, but you would have an empty dataset.

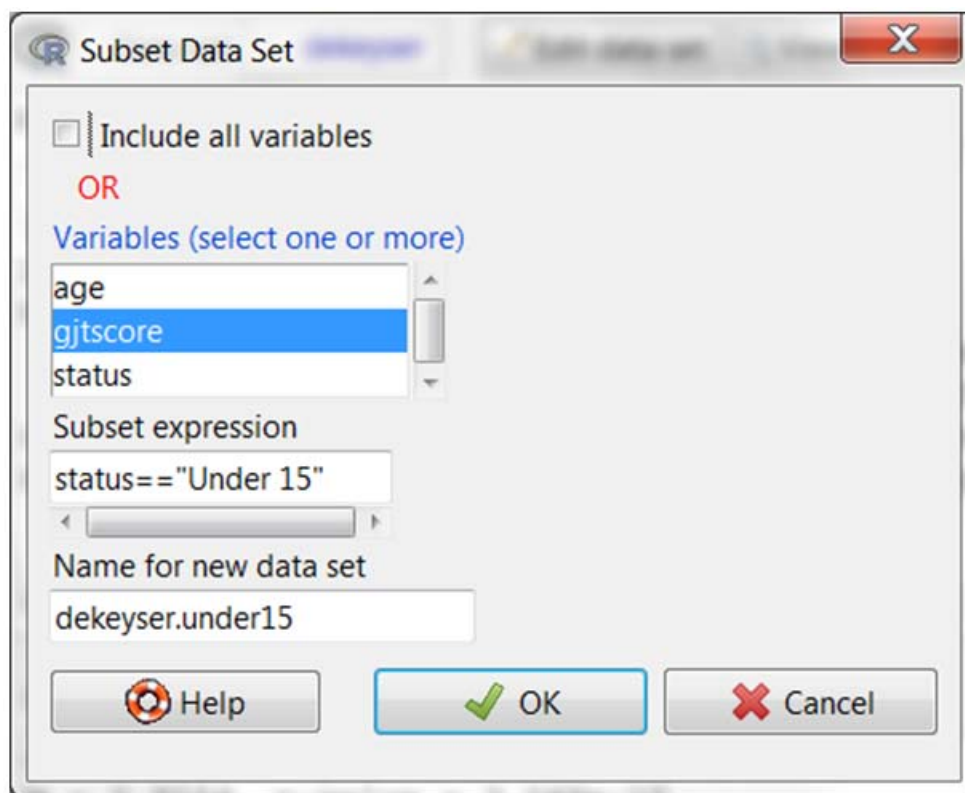


Figure 10 Subsetting a dataset in the long form (the dekeyser dataset)

To finish off the process, you’d need to go back and make the dekeyser file your active dataset, and repeat the process again, this time entering `status == "Over 15"` and naming the data file differently.

You can see that this process of splitting up your data into two files is a little time consuming and must be done exactly right. However, you can then continue to use all of the commands in R Commander. I think it is much easier to simply move to R console and specify exact row numbers if you have a command that doesn't split the file by groups. For example, Chapter 3 of the book shows a plot called a histogram, and in R Commander there is no way to call for histograms split by groups.

But in that case, you could simply open the "View dataset" button in R Commander, see which rows describe which groups (for the **dekeyser** dataset, rows 1 through 15 are the "Under 15" group, and rows 16–57 are the "Over 15" group), and type in the command for histograms with those row numbers. I myself would probably first run the menu sequence for the histogram in R Commander so I could get the R console command. For example, here is the command R Commander returns when I call for a histogram through the menus:

```
Hist(dekeyser$gjtscore, scale="frequency", breaks="Sturges", col="darkgray")
```

Now I simply add the row numbers in square brackets after copying the command over to the R console:

```
Hist(dekeyser$gjtscore[1:15], scale="frequency", breaks="Sturges",  
col="darkgray")
```

```
Hist(dekeyser$gjtscore[16:57], scale="frequency", breaks="Sturges",  
col="darkgray")
```

Bibliography

- DeKeyser, R. M. (2000). The robustness of critical period effects in second language acquisition. *Studies in Second Language Acquisition*, 22, 499–533.
- Everitt, B., & Dunn, G. (2001). *Applied multivariate data analysis* (2nd ed.). New York: Oxford University Press.
- Obarow, S. E. H. (2004). *The impact of music on the vocabulary acquisition of kindergarten and first grade students*. (Ed. Dissertation). Available from ProQuest Dissertations & Theses database. (UMI No. 3120733).
- Torres, J. (2004). *Speaking up! Adult ESL students' perceptions of native and non-native English speaking teachers*. Unpublished MA, University of North Texas, Denton.