The most powerful functions in Stata are `sort`, `reshape`, `collapse`, and `merge`. These functions are the bread and butter of Stata; and they justify the continued use of Stata in economic research. It seems like `R` is becoming more prevalent, however, as the types of analysis required for empirical research become more varied, and must rely more heavily on a wider developer base. It makes sense, then, to spend a little time learning how to replicate the `sort`, `reshape`, `collapse`, and `merge` capabilities in `R`.

We can first create a sample data frame for manipulation, and then print the output.

```
(X <- data.frame(id=c(1,1,2,2), t=c(1,2,1,2), x1=c(5,3,6,2), x2=c(6,5,1,4)))
```

```
  id t x1 x2
1  1 1  5  6
2  1 2  3  5
3  2 1  6  1
4  2 2  2  4
```

Sort

The difference between Stata and `R` is that, with Stata, the data are very rigid, which often simplifies operations for economic data. The data in `R`, however, can take many forms, including lists, vectors, and data frames. And there can be multiple data sets in memory simultaneously. Simple operations, then, must be more explicitly defined. Sorting data is not as simple as "sort data", but it's not too much more complicated. Instead, we arrange the data according to a list of indices.

First, as a refresher of indexing a data frame, suppose that we want to extract just the second row of X:

```
X[2, ]
```

```
  id t x1 x2
2  1 2  3  5
```

And the second through fourth rows:

```
X[2:4, ]
```

```
  id t x1 x2
2  1 2  3  5
3  2 1  6  1
4  2 2  2  4
```

And then, suppose, that we want to collect the second and fourth rows, except we ant to reverse the order:

```
X[c(4,2), ]
```

```
  id t x1 x2
4  2 2  2  4
2  1 2  3  5
```

Sorting is just a simple extension. If we collect the indices of an ordered variable, then we can sort the entire data frame. The `order` function facilitates this index ordering.

```
order(X$t)
```

```
[1] 1 3 2 4
```

The sorted data can be achieved by appropriately applying the ordered indices.

```
X[order(X$t), ]
```

```
  id t x1 x2
1  1 1  5  6
3  2 1  6  1
2  1 2  3  5
4  2 2  2  4
```

Reshape

The Stata reshape command has two basic options: wide and long. Suppose that a variable x is indexed along two dimensions, i and j. In wide format, each observation is uniquely identified by the i index, and the j indexed attributes are separate variables or columns. In long format, each observation is uniquely identified by a different i, j combination. All else equal, there will be more observations or rows in long format, and more variables or columns in wide format. Consider for example the data frame X, and let `id` be indexed by i and `t` be indexed by j. The data frame is therefore already in long format, as each observation is identified by an ID *and* and time period: individual 1 in period 1 is represented as a different observation than individual 1 in period 2. Suppose that we want the unit of observation to be only the unique IDs, converting the data frame to wide format. There are many, many ways to do this in `R`, but we will use the `reshape` command:

```
(wide <- reshape(X, idvar="id", timevar="t", direction="wide"))
```

```
  id x1.1 x2.1 x1.2 x2.2
1  1    5    6    3    5
3  2    6    1    2    4
```

Note that the arguments in the `reshape` command indicate that the natural use case is for panel data, where i indexes the unit ID variable and j indexes the time variable. Reshaping data is actually pretty fun — it's like playing the Creator (see Figure 1). Even for a simple data frame like `X`, there are $2^3 = 8$ different formats, since we can view the index on the `x` characteristic as yet another index. With respect to that index, then, the data frame `X` is actually in wide format. Consider the data frame where each observation represents a different combindation of the three indices, where the third is the 1 or 2 that follows the `x` variable. Instead of reshape, we can "melt" the data to long format:

```
(llong <- melt(X, id=c("id","t")))
```

```
  id t variable value
1  1 1       x1     5
2  1 2       x1     3
3  2 1       x1     6
4  2 2       x1     2
5  1 1       x2     6
6  1 2       x2     5
7  2 1       x2     1
8  2 2       x2     4
```

So far, we've seen the same data frame `X` with dimension 4×4, 2×5, and 8×4 — all representing the exact same information. Each representation may be useful in a different context.

Collapse

Suppose that we want to transform the data frame to reflect the average value of x_1 and x_2 across time periods $t \in \{1, 2\}$. In Stata, we'd use the `collapse` command; the comparable command, both in function and in name, within `R` is `aggregate`.

Calvin and Hobbes by Watterson
First there was nothing...
...then there was Calvin!
Calvin, the mighty god, creates the universe with pure will!
From utter nothingness comes swirling form! Life begins where once was void!
But Calvin is no kind and loving god! He's one of the old gods! He demands sacrifice!
Yes, Calvin is a god of the underworld! And the puny inhabitants of earth displease him!
The great Calvin ignores their pleas for mercy and the doomed writhe in agony!
HAVE YOU SEEN HOW ABSORBED CALVIN IS WITH THOSE TINKERTOYS? HE'S CREATING WHOLE WORLDS OVER THERE!
I'LL BET HE GROWS UP TO BE AN ARCHITECT.

Figure 1: Playing the creator with `reshape`

```
(aggdata <-aggregate(X, by=list(X$id), FUN=mean))
```

```
  Group.1 id   t x1  x2
1       1  1 1.5  4 5.5
2       2  2 1.5  4 2.5
```

The `Group.1` and `t` variables are redundant, here, for this very simple data set. The `aggregate` function supports much more complicated queries, however, and adding the group variables becomes useful. A common query in Stata is to collapse the data by multiple variables. The extension for `aggregate` is clear, since the `by` parameter is a list.

Merge

Consider another data frame `Y` with another characteristic for each unit and time period.

```
(Y <- data.frame(id=c(1,1,2,2), t=c(1,2,1,2), x3=c(4,3,4,3)))
```

```
  id t x3
1  1 1  4
2  1 2  3
3  2 1  4
4  2 2  3
```

To merge `X` and `Y` into a single data frame, we use the `merge` function. This is comparable to Stata's merge command, except that in `R`, you can have two data frames stored in memory; you do not need to save one to disk. Additionally, in `R`, the data sets do not need to be sorted before the merge. Together, these enhancements save many lines of code.

```
merge(X, Y, by=c("id", "t"))
```

```
  id t x1 x2 x3
1  1 1  5  6  4
2  1 2  3  5  3
3  2 1  6  1  4
4  2 2  2  4  3
```

A convenient aspect of `merge` in `R` is that, unlike Stata, the basis variables need not be named the same. Instead of a generic `by` parameter, we can specify the merging variables with `by.x` and `by.y`, which refer to the first and second data frames, respectively.

```
names(Y) <- c("a", "b", "x3")
merge(X, Y, by.x=c("id", "t"), by.y=c("a", "b"))
```

```
  id t x1 x2 x3
1  1 1  5  6  4
2  1 2  3  5  3
3  2 1  6  1  4
4  2 2  2  4  3
```

Merges in economic data, especially panel data, are often used to attribute static characteristics to the time series. The target data may be organized by time and unit, whereas the new data frame may be at just the unit level. The two data frames need not ber perfectly aligned, as in the previous examples. Consider a new, unit-level data frame `Z` that contains static characteristics (that do not depend on time).

```
(Z <- data.frame(id=c(1,2), x4=c("yes", "no")))
```

```
  id  x4
1  1 yes
2  2  no
```

We can merge this into the panel data frame X using roughly the same syntax.

```
merge(X, Z, by=c("id"))
```

```
  id t x1 x2  x4
1  1 1  5  6 yes
2  1 2  3  5 yes
3  2 1  6  1  no
4  2 2  2  4  no
```