This section will go through some basic data manipulation commands in R, commands that will hopefully be immediately useful to you as you work through the first problem set. I will cover sorting, reshaping, collapsing, and merging data. As always, see the R help file or the list of R resources in the section syllabus for further exploration of these commands.

R for Stata users

Before we begin, it is worth noting that Stata excels at these kinds of tasks, and for that reason has been the primary tool for empirical work in economics. Anecdotally, this seems to be shifting somewhat for a variety of reasons, and it may very well be that ten years from now the majority of the empirical work we do will be R. Or not.

I don't have a strong desire to get into the "R vs. Stata" debate, but I will say that at some point in your career you will probably need to be conversant in both. Since you will definitely need to learn R for this course, all that really means is that you will probably also use Stata, probably when working with a professor or with a coauthor who prefers Stata. In my own work, I use Stata, R, and occasionally Matlab, depending on the need and on my colleagues' preferences.

I'm going to end this pointless digression with a link to a pdf that compares R and Stata more thoroughly. Those of you with a Stata background may find it useful, while the rest of you won't care.

Preliminaries

First we create a simple data frame X with four observations and four variables named id, t, w, and y.

```
(X <- data.frame(id=c(1,1,2,2), t=c(1,2,1,2), w=c(5,3,7,2), y=c("A","B","A","B")))
```

```
  id t w y
1  1 1 5 A
2  1 2 3 B
3  2 1 7 A
4  2 2 2 B
```

All of the variables except for y are numeric, while y is what R calls a "factor variable," which is basically a categorical variable. By the way, the class() command is useful for figuring out what kind of variables you're working with.

```
c(class(X$w),class(X$y))
```

```
[1] "numeric" "factor"
```

The $ above lets us refer to variables inside the data frame X — it's the equivalent of a . in other languages. Think of X$w as "the variable named w within the data frame X".

Sorting data

Now that we've got a data frame, we'd like to sort it. To do this we'll rearrange the data using a set of indices that happen to be the order we want. How does this work? Suppose we wanted to sort the data on our `w` variable (with lower numbers at the top). With such a small dataset, we could do this manually, noting that the true order of the rows should be 4, 2, 1, 3:

```
X[c(4,2,1,3), ]
```

```
  id t w y
4  2 2 2 B
2  1 2 3 B
1  1 1 5 A
3  2 1 7 A
```

It's important to include the comma to tell `R` that this is the order of *rows* that we want, otherwise it will rearrange the columns instead!

We can see that this worked, but it's not really a generalizable solution: eyeball-sorting on large datasets seems like a pretty boring way to spend an afternoon. Fortunately `R` has a function called `order()` that figures out the proper order of the rows for us:

```
order(X$w)
```

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

Conveniently, `order()` has returned a vector with the index order we came up with manually, so all we have to do to put it together is the following:

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

```
  id t w y
4  2 2 2 B
2  1 2 3 B
1  1 1 5 A
3  2 1 7 A
```

And we're done!

Reshaping data

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 again 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 individual identifier 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 w.1 y.1 w.2 y.2
1  1   5   A   3   B
3  2   7   A   2   B
```

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. There are many other ways to view the data. For example, each observation could be a period in time. In fact, there are $\$2^3 = 8\$$[1] different ways we could rearrange this data. Consider the data frame where each observation represents a different combination of the three indices, with the third "index" being either w or y. Instead of reshape, we can "melt" the data to long format:

```
library(reshape)
(long <- melt(X, id=c("id","t")))
```

```
Error in library(reshape) : there is no package called reshape
Error: could not find function "melt"
```

Of course, all of these are representations of the exact same data. Each may be useful in a different context. Moreover, datasets come packaged in a wide variety of wide and long formats[2], so wrapping your head around the `reshape()` and `melt()` commands will be well worth your while.

Collapsing data

Now suppose that we want to transform the data frame to reflect the average value of w 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`.

```
(aggregate(X$w, by=list(X$id), FUN=mean))
```

```
  Group.1   x
1       1 4.0
2       2 4.5
```

Here, we're returning the average of `w` by `id`. Note that the `by` argument expects a list, even if that list contains only one item.

Merging data

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

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

```
  id t x
1  1 1 4
2  1 2 3
3  2 1 4
4  2 2 3
```

[1] This math to this is interesting to play around but it's probably not a useful exercise

[2] For example, many forms of census data.

Let's say we want to merge the data in `X` and `Y` into a single data frame using the `id` and `t` variables as our unique identifiers:

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

```
  id t w y x
1  1 1 5 A 4
2  1 2 3 B 3
3  2 1 7 A 4
4  2 2 2 B 3
```

This is actually a great deal easier than in Stata. First, since `R` can keep multiple datasets in memory we don't need to save one to disk. Second, the variables do not need to have the same names in order to sort them:

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

```
  id t w y x3
1  1 1 5 A  4
2  1 2 3 B  3
3  2 1 7 A  4
4  2 2 2 B  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 the same syntax.

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

```
  id t w y  x4
1  1 1 5 A yes
2  1 2 3 B yes
3  2 1 7 A  no
4  2 2 2 B  no
```

In Stata, this would be called a 1:m merge, since observations in `X` are matched to multiple observations in `Z`. `R` does not require us to make that distinction.