This section will explore some of the real-world issues involved with performing econometric analyses. In the real world, data is messy. It comes in weird formats, has unusual representation for missing values, and contains errors of every kind imaginable. The process of turning messy, poorly structured data into cleaned data ready for analysis is called **data cleaning**, and while it may not seem important now, you can believe me when I tell you that you will spend a healthy proportion of your post-coursework time in graduate school cleaning data. The purpose of this section is to ease your introduction into that process by providing you with a few basic tools here.

To illustrate, we'll work through a real-life research question that I developed some time ago. We'll be investigating whether individuals' belief in climate change is affected by changes in their states' reliance on carbon-intensive industry. Our goal is to construct a dataset in which we can run following regression:

$$\text{CC Belief}_{ist} = \alpha + \text{Oil/Gas Employment}_{st} + \text{Covariates}_{st} + \varepsilon_{ist}$$

where i represents an individual, s their state, and t is some time period.

Loading the data

To measure our dependent variable, an individual's belief in climate change, we'll use a series of Pew Surveys[1] that asked randomly selected individuals a series of questions about their beliefs in global warming. We'll be primarily interested in their answer to the following:
From what you¡¯ve read and heard, is there solid evidence that the average temperature on earth has been getting warmer over the past few decades, or not?
When I did this project, I actually loaded in data from 8 surveys. We'll focus on just two, one from November 2011 and the other from October 2012. After downloading them from the appropriate spot on the Pew website[2], we load the October 2012 data into `R`.

```
rm(list = ls())
oct12.df <- read.table("EarlyOct12 public.csv", header = T, sep = ",")
head(names(oct12.df))
```

```
[1] "X"        "psraid"   "sample"   "int_date" "fcall"    "attempt"
```

Since we only care about a small subset of the variables reported on in the survey, we'll keep those and drop the rest. We can do by requesting either certain column numbers or certain variable names; here, we'll use variable names. Since the variable names aren't very informative, we'll need the documentation to figure out what is what. I used `Early Oct12 que.public.docx` to figure out that we want questions 61, 62, and 63.

```
oct12.df <- oct12.df[c("int_date","state","q61","q62","q63")]
head(oct12.df, n = 1)
```

[1] Available from `http://www.people-press.org/question-search/`.
[2] Surprisingly non-trivial, but this isn't a course on navigating websites so I'll skip over that.

```
  int_date         state q61                  q62
1   100412 Massachusetts Yes Very serious problem
                                                                                  q63
1 Yes, scientists generally agree that the earth is getting warmer because of human activity
```

Finding errors

Great! We've got some data. When loading any new dataset, however, we should be wary of errors. In this case, we have very lengthy question answers. We might be worried that these long character strings are encoded with typos for some observations. To look into this, wWe'll want to code these long character strings as factor variables, which we covered last time. Factorizing variables has the added benefit of reducing memory usage and making analysis easier. Win-win-win!

```
oct12.df$q1f <- factor(oct12.df$q61)
oct12.df$q2f <- factor(oct12.df$q62)
oct12.df$q3f <- factor(oct12.df$q63)
```

We can examine the levels of the different factors using the `levels` command. The levels of a factor variable are just the number of different values this variable assumes in the data.

```
levels(oct12.df$q1f)
```

```
[1] "(VOL.) Don't know/Refused"  "(VOL.) Mixd/some evidence"
[3] "(VOL.) Mixed/some evidence" "No"
[5] "Yes"
```

Uh oh! It looks like `q1f` has an error somewhere — something was misentered[3] and now we have two redundant factors. An easy way to fix this is to fix the original character variable, `q61`, and then make it into a factor again.

```
badindices <- which(oct12.df$q61 == c("(VOL.) Mixd/some evidence"))
oct12.df$q61[badindices] <- "(VOL.) Mixed/some evidence"
```

This is a good time to talk about the `which` command, which I just used. `which` gives the `TRUE` indices of a logical object, letting us locate the errors. In this case, there is only one, and we can fix it by setting the correct string. Is it fixed?

```
oct12.df$q1f <- factor(oct12.df$q61)
levels(oct12.df$q1f)
```

```
[1] "(VOL.) Don't know/Refused"  "(VOL.) Mixed/some evidence"
[3] "No"                         "Yes"
```

It's fixed! Now let's load November 2011 survey data and format it similarly.

```
nov11.df <- read.table("Nov11 public.csv", header = T, sep = ",")
```

[3] In fairness to the Pew people, I created this error. Their data is unusually clean.

But we have to be careful here, since the numbers of the relevant questions have changed. We again turn to the documentation. This is why you should always save documentation when you download data! Don't wait to get it later. You'll forget where it is.

```
nov11.df <- nov11.df[c("int_date","state","q65", "q66", "q67")]
nov11.df$q1f <- factor(nov11.df$q65)
nov11.df$q2f <- factor(nov11.df$q66)
nov11.df$q3f <- factor(nov11.df$q67)
```

Notice that I've conviently named the factor variables to match `oct12.df`. This is not by accident, since we ultimate want to join the two data frames together. But first, we need to check the levels of our factor variables again. Are they the same?

```
all(levels(nov11.df$q1f) == levels(oct12.df$q1f))
levels(nov11.df$q1f)
levels(oct12.df$q1f)
```

```
[1] FALSE
[1] "Dont know/Refused (VOL.)"  "Mixed/some evidence (VOL.)"
[3] "No"                         "Yes"
[1] "(VOL.) Don't know/Refused"  "(VOL.) Mixed/some evidence"
[3] "No"                         "Yes"
```

Crap, they don't match. This is a dumb problem. Fortunately, since they're in the same order, it's easy to fix.

```
levels(nov11.df$q1f) <- levels(oct12.df$q1f)
```

Now, let's put these data frames together! We'll use `rbind`.

```
data.df <- rbind(nov11.df, oct12.df)
```

```
Error in match.names(clabs, names(xi)) :
  names do not match previous names
```

Crap again. The names don't match because the question numbers don't match. But, we set the factor variables names so that they **would** match! Since the factor variables contain all of the information in the character variables, let's only use the well-named factor variables.

```
data.df <- rbind(nov11.df[, c(1, 2, 6, 7, 8)], oct12.df[, c(1, 2, 6, 7, 8)])
head(data.df, n = 3)
```

```
  int_date        state                        q1f  q2f             q3f
1   110911     Michigan                         No <NA> Somewhat serious
2   110911 Pennsylvania (VOL.) Mixed/some evidence <NA>     Very serious
3   110911     Maryland  (VOL.) Don't know/Refused <NA>     Very serious
```

Dealing with dates

Great! We've got a dataset. We have the date of the interview, the state of the respondent, and their answers to the survey questions. Let's take a closer look at our `int_date` variable. What kind of variable is it?

```
class(data.df$int_date)
```

```
[1] "integer"
```

It's an integer. This is okay, but it's not ideal. Often, we'll want to use a particular type of variable to store dates — this is often more efficient, memory-wise, and it enables us to use a wide variety of date-specific functions. To turn `int_date` into a date variable, we'll first turn it into a character string and then use the function `as.Date` to convert it to a date that `R` will recognize.

```
data.df$int_date.str <- as.character(data.df$int_date)
data.df$int_date.date <- as.Date(data.df$int_date.str, format = "%y%d%m")
class(data.df$int_date.date)
```

```
[1] "Date"
```

That's it! We now have a functioning date variable. This is useful for many reasons. In particular, we can now use this to determine on what day of the week the survey occured.

```
dow <- weekdays(data.df$int_date.date)
head(dow)
```

```
[1] "Wednesday" "Wednesday" "Wednesday" "Wednesday" "Saturday"  "Monday"
```

Dropping, renaming, and reording variables

Often while cleaning, we'll end up with extraneous variables that we created in the process of making other variables. We could just leave them be, but if we're working with a lot of data or if we are just a tiny bit obsessive about not letting our environment get flooded with objects[4] we might want to clean it up every now and again. We also often want to rename and reorder variables within a data frame. Fortunately, `R` makes all of this easy. First, we'll drop our extraneous string and integer variables from our data frame. Then, we'll rename the awkwardly entitled `data.df$int_date.date` to just `data.df$int_date`. Finally, we'll reorder our data with the date at the beginning.

```
data.df$int_date.str <- data.df$int_date <- NULL
names(data.df)[names(data.df) == "int_date.date"] <- "int.date"
data.df <- data.df[c(5,1:4)]
```

[4] Guilty!

Melting and reshaping data

Initial graphs

Multiple histograms, boxplots

**

Future sections

Since only four sections remain after this one, I want to solicit your feedback on what you'd like to learn in the remaining section. At present, my plan is to do the following:

Section 10: Instrumental Variables
Section 11: Web scraping in `R`
Section 12: Spatial analysis
Section 13: Using and analyzing satellite imagery

I'm not married to this itinerary, however. You should note that only section 10 will be directly pertinent to what Max is covering in lecture — the others are topics I've included due to personal interest. If there is a strong desire to cover other topics (or to do a general review section), I'm happy to change this plan.