In the real world, data is messy. The process of turning messy, poorly structured data into cleaned data ready for analysis is called **data cleaning**. Those of us who aren't yet fancy enough to hire research assistants can expect to do a great deal of this in the initial stages of research. It can be frustrating, so 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 state-level belief in climate change is affected by changes in the presence of carbon-intensive industries. Our goal is to construct a dataset in which we can run following regression:

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

where i represents an individual, s their state, and t is some time period. Actually, we'll end up running a somewhat different regression, but this gives you an idea of where we're headed.

Loading the data

To measure our dependent variable, 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. This is done 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.

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

```
  int_date         state q61
1   100412 Massachusetts Yes
```

[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.

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. One way to look into this is 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)
```

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] "No"                          "(VOL.) Don't know/Refused"
[3] "(VOL.) Mixd/some evidence"   "(VOL.) Mixed/some evidence"
[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 change 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] "No"                          "(VOL.) Don't know/Refused"
[3] "(VOL.) Mixed/some evidence" "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 = ",")
```

But we have to be careful here, since the number of the relevant question has 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")]
nov11.df$q1f <- factor(nov11.df$q65)
```

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?

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

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

```
[1] FALSE
[1] "Don\x92t know/Refused (VOL.)" "Mixed/some evidence (VOL.)"
[3] "No"                           "Yes"
[1] "No"                          "(VOL.) Don't know/Refused"
[3] "(VOL.) Mixed/some evidence" "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) <- c("Don't know","Mixed","No","Yes")
```

Note that we also simplied the factor levels. 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 original question variable names 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, 4)], oct12.df[, c(1, 2, 4)])
head(data.df, n = 3)
```

```
  int_date        state        q1f
1   110911     Michigan         No
2   110911 Pennsylvania      Mixed
3   110911     Maryland Don't know
```

Dealing with dates

Great! Getting there. 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; for example, we can now the `weekdays` command 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(3, 1, 2)]
head(data.df)
```

```
    int.date        state        q1f
1 2011-11-09     Michigan         No
2 2011-11-09 Pennsylvania      Mixed
3 2011-11-09     Maryland Don't know
4 2011-11-09     New York        Yes
5 2011-11-12  Connecticut        Yes
6 2011-11-14      Florida        Yes
```

Collapsing data

We've got our dependent variables — in fact, we have a few of them. Now we want state employment by year we can get these from the Bureau of Labor Statistics. The ones I found are actually a set of excel files that require some additional formatting to use in our setting. However, to save time and sanity[5], I've skipped some setps here and created `oilgas_emp.csv`, which we'll import now. Party on, Wayne.

[4] Guilty!

[5] Importing excel files in `R` is actually non-trivial and requires either Perl, Java, or an OS-specific package. None of these are easy to present in section, so I'll omit them now. See the following link for help on importing excel files: `http://tinyurl.com/m9pbz8v`.

```
oilgas.df <- read.table("oilgas_emp.csv", header = T, sep = ",")
head(oilgas.df)
```

```
  year month oilgas_emp   N   state
1 2003     1        676 657 ALABAMA
2 2003     2        676 657 ALABAMA
3 2003     3        676 657 ALABAMA
4 2003     4        665 657 ALABAMA
5 2003     5        662 657 ALABAMA
6 2003     6        657 657 ALABAMA
```

Party on, Garth. We'll just focus on the variable `N`, which is the annual employment level for each state-year. This means that we can drop all of the extraneous observations that list the monthly employment, a process which is called collapsing the data. `Stata` users will recall the command `collapse` which performs this function; `R` does it using `aggregate`, among other methods.

```
oilgas.yearly <- aggregate(oilgas.df$N, by = list(oilgas.df$year, oilgas.df$state), FUN = mean)
names(oilgas.yearly) <- c("year", "state", "emp")
```

`aggregate` takes in the variable we want to aggregate over, a list of grouping elements, and the function to apply to each subset. We use `mean` since we're taking the mean over the year of an annual statistic, which returns... the annual statistic. A little silly, but it works. After renaming our new data frame appropriately, we've got something we can merge to our `data.df` data frame!

Merging data

Before merging, we need to make sure that the values in our variables match up. There are two things we need to fix. First, the `state` in `oilgas.yearly` is upper-case, while the `state` in `data.df` is not. Let's fix that.

```
data.df$state <- toupper(data.df$state)
```

Since we're matching on year, we also need a `year` variable in our `data.df`. There are a few ways to do this, but the way I find most intuitive is to use the `format` function on our date variable and then convert it to `numeric`.

```
data.df$year <- as.numeric(format(data.df$int.date, "%Y"))
```

Great! Now, we're ready to merge. We've conveniently named our variables so that they match in both data frames, but this isn't strictly necessary in `R`. It just makes life easier.

```
merged.df <- merge(x = data.df, y = oilgas.yearly, by = c("year", "state"), all.x = T)
head(merged.df)
```

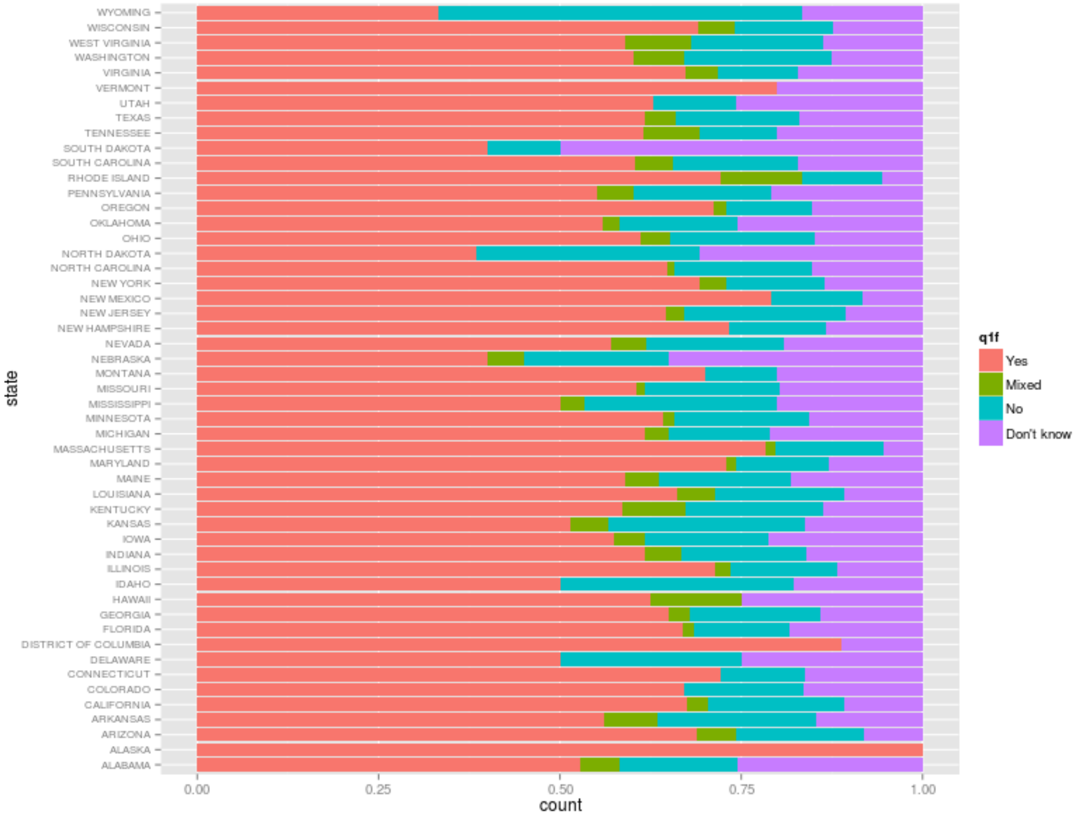
```
  year   state   int.date        q1f emp
1 2010 ALABAMA 2010-12-07      Mixed 779
2 2010 ALABAMA 2010-12-04        Yes 779
3 2010 ALABAMA 2010-12-07        Yes 779
4 2010 ALABAMA 2010-12-06        Yes 779
5 2010 ALABAMA 2010-12-05 Don’t know 779
6 2010 ALABAMA 2010-12-06 Don’t know 779
```

The first two arguments are our two date frames, `by` specifies a vector of variables that we want to merge on, and `all.x = T` indicates to `R` that we should keep all of the observations from our first dataset, `data.df`, even if they don't have a match in `oilgas.yearly`.

A neat graph

What can we do with this dataset? Let's do a little plotting, to start. We haven't worked with factor (categorical) variables much before, so this is a good opportunity to practice graphing them. What if we want to see the proportion of responses by state? Again, we'll load and use `ggplot2`.

```
library(ggplot2)
merged.df$q1f <- factor(merged.df$q1f, levels = c("Yes", "Mixed", "No", "Don’t know"))
ggplot(merged.df, aes(x = state, fill = q1f)) + geom_bar(position = "fill") +
  coord_flip() + theme(axis.text.y=element_text(size=rel(0.8)))
```



There are lots of things we might like to improve about this graph, starting with the order of the states, but it's a nice visual.

Running regressions

We're almost to the point where we can run a regression with our data. However, we need to make some decisions first.

First, how should we encode `q1f`: should we treat it as an ordered variable, with increasing levels of certainty in global warming encoded as a higher number? Or should we simply code it as 1 if yes and 0 otherwise? There's no right answer here, it just depends on our research design. For ease of demonstration, we'll follow the binary route.

Second, what is the appropriate level of observation for our regression? The natural choice would be to run OLS with each response as an observation, but this is actually not a good idea. First because OLS is not appropriate when our dependent variable is binary[6], and second because our independent variable, oil and gas employment by state-year, does not vary for individuals within a given state-year. So if we don't cluster our errors appropriately, we may overstate our statistical power when calculating test statistics.

Since we haven't learned about logistic regression (the solution to having a binary dependent variable) or clustering, the easiest option is to run OLS on the average response by state-year. Once again, we'll use the aggregate command.

```
merged.df$q1b <- ifelse(merged.df$q1f == "Yes", 1, 0)
bystate.df <- aggregate(cbind(merged.df$q1b,merged.df$emp),
  by = list(merged.df$year, merged.df$state), FUN = mean)
names(bystate.df) <- c("year","state","yes","emp")
tail(bystate.df)
```

```
    year         state       yes  emp
97  2010 WEST VIRGINIA 0.5714286 2244
98  2011 WEST VIRGINIA 0.6000000 2181
99  2010     WISCONSIN 0.7407407   NA
100 2011     WISCONSIN 0.6666667   NA
101 2010       WYOMING 0.0000000 4197
102 2011       WYOMING 0.4000000 4316
```

By looking at the tail of our data we can see that some values are missing from emp. We clearly don't want to include these in our regression, so we'll need to drop them. The easiest way to do this is to use is.na.

```
bystate.df <- bystate.df[!is.na(bystate.df$emp), ]
tail(bystate.df)
```

```
    year         state       yes  emp
93  2010      VIRGINIA 0.6538462  446
94  2011      VIRGINIA 0.6896552  445
97  2010 WEST VIRGINIA 0.5714286 2244
98  2011 WEST VIRGINIA 0.6000000 2181
101 2010       WYOMING 0.0000000 4197
102 2011       WYOMING 0.4000000 4316
```

Okay! We're ready to run OLS.

```
OLS <- function(y,X) {
  n <- nrow(X); k <- ncol(X)
  b <- solve(t(X) %*% X) %*% t(X) %*% y
  e <- y - X %*% b; s2 <- t(e) %*% e / (n - k); XpXinv <- solve(t(X) %*% X)
  se <- sqrt(s2 * diag(XpXinv))
  t <- b / se
  p <- 2 * pt(-abs(t),n-k)
```

[6] For more on this see Greene chapter 17.

```
  output <- data.frame(b, se, t, p)
  colnames(output) <- c("Estimate","Std. Error", "t statistic", "p-value")
  return(output)
}
y <- bystate.df$yes
X <- cbind(1, bystate.df$emp)
(output <- OLS(y,X))
```

```
      Estimate   Std. Error t statistic      p-value
1 6.028694e-01 1.947178e-02  30.9611805 1.388631e-42
2 2.488441e-07 1.311483e-06   0.1897425 8.500601e-01
```

Well, we didn't find anything. Darn. Of course, there are some really huge problems with this regression, at least the way we've set it up. Can you see them?

What else?

Cleaning data is almost always an adventure, since every project will present different and entirely unique challenges. At first, the cleaning process will seem interminably long and tedious, but you'll become faster over time. You'll develop habits of research that enable to get you to your end goal as quickly as possible. I couldn't possibly cover everything you'll need to know to clean your next dataset, but I hope to have laid at least a bit of a blueprint. In my own research, I've found it valuable to spend some time thinking about exactly what regression I want to run so that I always have an idea of where I'm headed. It's important to keep the goal in mind when cleaning so that you don't run down too many rabbit holes before you're even sure if the project is worth pursuing. If it's not mission critical for the basic question that you want to ask, make a note to yourself, move on, and return to it later.

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.