Editing survey data (or, how to deal with pregnant men)

Ben Klemens

7 June 2015

This post is partly a package announcement for Tea, a package for editing and imputation[1]. But it mostly discusses the problem of editing, because I've found that a lot of people put no thought into this important step in data analysis.

If your data is a survey filled in by humans, or involves using sensors that humans operated, then your data set will have bad data.

But B, you object, *I'm a downstream user of a data set provided by somebody else, and it's a clean set—there are no pregnant men—so why worry about editing?* Because if your data is that clean, the provider already edited the data, and may or may not have told you what choices were made. Does nobody have gay parents because nobody surveyed has gay parents, or because somebody saw those responses and decided they must be an error? Part of the intent of Tea is to make it easy to share the specification of how the data was edited and missing data imputed, so end users can decide for themselves whether the edits make sense.

If you want to jump to working with Tea itself, start with the Tutorial/manual[2].

For editing bad values in accounting-type situations, you may be able to calculate which number in a list of numbers is incorrect—look up *Fellegi-Holt imputation* for details. But the situations where something can be derived like this are infrequent outside of accounts-based surveys.

So people wing it.

Some people will *listwise delete* the record with some failure in it, including a record that is missing entirely. This loses information, and can create a million kinds of biases, because you've down-weighted the deleted observation to zero and thus up-weighted all other observations. An analyst who deletes the observations with N/As for question 1 only when analyzing question 1, then restores the data set and deletes the N/As for question 2 when analyzing question 2, ..., is creating a new weighting scheme for every question.

I don't want to push you to use Tea, but I will say this: listwise deletion, such as using the ubiquitous `na.rm` option in R functions, is almost always the wrong thing to do.

Major Census Bureau surveys (and the decennial census itself) tend to lean on a preference ordering for fields and deterministic fixes. Typically, age gets fixed first,

[1] `https://b-k.github.io/tea-tutorial`
[2] `http://b-k.github.io/tea-tutorial/`

comparing it to a host of other fields, and in a conflict, age gets modified. Then age is deemed edited, and if a later edit involving age comes up [it's complicated...], then age stays fixed and the other fields are modified. There is usually a deterministic rule that dictates what those modifications should be for each step. This is generally referred to as *sequential editing*.

Another alternative is to gather evidence from all the edits and see if some field is especially guilty. Maybe the pregnant man in our public health survey also reports using a diaphragm for contraception and getting infrequent pap smears. So of the pair (man, pregnant), it looks like (man) is failing several edits.

If we don't have a deterministic rule to change the declared-bad field, then all we can do is blank out the field and impute, using some model. Tea is set up to be maximally flexible about the model chosen. Lognormal (such as for incomes), Expectation-Maximization algorithm, simple hot deck draw from the nonmissing data, regression model—just set *method: lognormal* or *EM* or *hot deck* or *ols* in the spec file, and away you go.

Tea is set up to do either sequential or deterministic edits. Making this work was surprisingly complicated. Let me give you a worst-case example to show why.

E.g. Assume these edits: fail on¡br¿ (age $<$ 15 and status=married),¡br¿ (age $<$ 15 and school=PhD),¡br¿ and (age$>$65 and childAge $<$ 10 $=>$ change childAge to age-25).¡br¿ The third one has a deterministic if-then change attached; the first two are just edits.

I think these are reasonable edits to make. It is entirely possible for a 14 year old to get a PhD, just as there has existed a pregnant man[3]. Nonetheless, the odds are much greater when your survey data gives you a 12-year old PhD or a pregnant man that somebody committed an error, intentional or not.

Then, say that we have a record with (age=14, married, PhD, childAge=5), which goes through these steps:

- Run the record through the edits. Age fails two edits, so we will blank it out.
- Send age to the imputation system, which draws a new value for the now-blank field. Say that it drew 67.
- Run the record through the edits, and the deterministic edit hits: we have to change the child's age to 42.

First, this demonstrates why writing a coherent editing and imputation package is not a pleasant walk in the park, as edits trigger imputations which trigger edits, which may trigger more imputations.

Second, it advocates against deterministic edits, which can be a leading cause of these domino-effect outcomes that seem perverse when taken as a whole.

Third, it may advocate against automatic editing entirely. It's often the case that if we can see the record, we could guess what things should have been. It's very plausible that age should have been 41 instead of 14. But this is guesswork, and impossible to automate. From Census experience, I can say that what you get when you add up these little bits of manual wisdom for a few decades is not necessarily the best editing system.

[3] `https://en.wikipedia.org/wiki/Thomas_Beatie`

But not all is lost. We've generated consistent micro-data, so no cross-tabulations will show anomalies. If we've selected a plausible model to do the imputations, it is plausible that the aggregate data will have reasonable properties. In many cases, the microdata is too private to release anyway. So I've given you a worst-case example of editing gone too far, but even this has advantages over leaving the data as-is and telling the user to deal with it.

Multiply impute Imputation is the closely-allied problem of filling in values for missing data. This is a favored way to deal with missing data, if only because it solves the weighting problem. If 30% of Asian men in your sample didn't respond, but 10% of the Black women didn't respond, and your survey oversampled women by 40%, and you listwise delete the nonresponding observations, what reweighting should you do to a response by an Asian woman? The answer: side-step all of it by imputing values for the missing responses to produce a synthetic complete data set and not modifying the weights at all.

Yes, you've used an explicit model to impute the missing data—as opposed to the implicit model generated by removing the missing data. To make the model explicit is to face the reality that if you make any statements about a survey in which there is any missing data, then you are making statements about unobserved information, which can only be done via some model of that information. The best we can do is be explicit about the model we used—a far better choice than omitting the nonresponses and lying to the reader that the results are an accurate and objective measure of unobserved information.

We can also do the imputation multiple times and augment the within-imputation variance with the across-imputation variance that we'd get from different model-based imputations, to produce a more correct total estimate of our confidence in any given statistic. That is, using an explicit model also lets us state how much our confidence changes because of imputation.

There are packages to do multiple imputation without any editing, though the actual math is easy. Over in the other window, here's the R function I use to calculate total variance, given that we've already run several imputations and stored each (record,field,draw) combination in a fill-in table. The `checkOutImpute` function completes a data set using the fill-ins from a single imputation, generated by the imputation routine earlier in the code. There's, like, one line of actual math in there.

```
library(tea)

get_total_variance <- function(con, tab, col, filltab, draw_ct, statvar){
    v <- 0 #declare v and m
    m <- 0
    try (dbGetQuery(con, "drop table if exists tt"))
    for (i in 1:draw_ct){
        checkOutImpute(dest="tt", origin=tab, filltab=filltab, imputation_number=i-1)
        column <- dbGetQuery(con, paste("select ", col, " from tt"))
        vec <- as.numeric(column[,1]) #type conversions.
        v[i] <- statvar(vec)
        m[i] <- mean(vec)
```

```
    }
    total_var <- mean(v) + var(m)/(1+1./draw_ct)
    return(c(mean(m), sqrt(total_var)))
}

# Here's the kind of thing we'd use as an input statistic: the variance of a Binomial
binom_var <- function(vec){
    p = mean(vec)
    return(p*(1-p)/length(vec))
}
```