Analyzing and Pooling Results From Multiply Imputed Data

David Disabato

2/13/2021

Multiple imputation

Multiple imputation is a statistical technique for handling missing data. It outperforms classical approaches to treating missing data, such as listwise deletion or mean imputation by generating more unbiased parameter estimates and more efficient standard errors. Multiple imputation takes a dataset with missing values and creates m complete datasets that can be analyzed with complete data analytic approaches.

In this blog post, I will be explaining analyzing multiply imputed data and pooling the results. What I was first learning about multiple imputation, one of the most confusing part was how to combine the results from each imputed dataset. I had m sets of results rather than just 1. Which one was correct? How did I get a single "final" result? That is what we are going to be going over today.

The main thing to keep in mind about analyzing multiply imputed data is that you always do the analysis - or extract the statistical information - separately for each imputed dataset. Then you **pool** (aka combine) the results from each analysis together with specific formulas. Two things you **DON'T** do is 1) only analyze one of the imputed datasets and use that as the final result or 2) stack the imputed datasets together and analyze that "super" dataset to get the final result [1].

For this blog post, I am assuming we have already created our multiply imputed datasets. Here, we will be just going over what to do once we have a set of multiply imputed datasets and wish to conduct our analysis. I show how I create the imputed data for readers interested in the code I used to setup the blog post, but I will not be discussing those details. For the statistical programming, I will be using R - open source computer software. For more information about R go to https://www.r-project.org/about.html.

I will also be using a package that I created called `str2str` (read as "structure to structure"), which contains a lot of simple wrapper functions for converting R objects from one structure to another. I find using these functions save a few lines of code and generally makes code easier to read. If you want to learn more about the package, you can go to the str2str documentation webpage.

Creating the multiply imputed data

Let's start by adding some missing values to a dataset.

```
suppressPackageStartupMessages(library(mice))
set.seed(01042021) # set seed so the imputed data are the same every time
ampute_obj <- ampute(data = attitude, prop = .50, bycases = TRUE)
attitude_na <- ampute_obj[["amp"]]
print(attitude_na)
```

[1] For some types of analyses, a "supermatrix" dataset can be used to get unbiased parameter estimates. The difficulty is obtaining correct standard errors and/or confidence intervals. See Lang & Little (2014) for a creative example.

```
##    rating complaints privileges learning raises critical advance
## 1      NA         51         30       39     61       92      45
## 2      63         64         51       54     63       73      47
## 3      71         NA         68       69     76       86      48
## 4      61         63         45       47     54       84      35
## 5      81         78         56       66     NA       83      47
## 6      43         55         49       44     54       49      34
## 7      58         67         42       56     66       68      35
## 8      71         75         50       55     70       66      41
## 9      72         82         72       67     71       83      31
## 10     67         61         NA       47     62       80      41
## 11     64         53         53       58     58       NA      34
## 12     67         NA         47       39     59       74      41
## 13     69         62         57       42     55       63      NA
## 14     68         NA         83       45     59       77      35
## 15     77         77         54       72     79       NA      46
## 16     81         90         50       72     60       54      36
## 17     74         85         64       NA     79       79      63
## 18     65         60         65       75     55       80      60
## 19     65         70         46       57     75       85      46
## 20     50         58         68       54     64       78      52
## 21     50         40         33       34     43       NA      33
## 22     64         61         52       62     66       80      41
## 23     53         66         52       50     63       80      37
## 24     40         37         42       58     50       57      49
## 25     63         54         42       48     66       75      33
## 26     66         77         66       63     NA       76      72
## 27     78         75         58       74     80       NA      49
## 28     48         57         44       45     51       83      38
## 29     85         85         71       NA     77       74      55
## 30     82         82         39       59     64       78      39
```

Here is the dataset with missing values we will be working off: `attitude_na`. It is simply the `attitude` data.frame in the default R package `datasets` with added missing values. The scores are in Percentage of Maximum Possible (POMP) units (Cohen, Cohen, Aiken, & West, 1999). Let's first see how much missing data we have.

```
library(str2str)
tmp <- lapply(X = attitude_na, FUN = function(x) sum(is.na(x)) / length(x))
miss_byvrb <- lv2v(tmp, use.vecnames = FALSE)
print(miss_byvrb)
```

```
##     rating complaints privileges   learning     raises   critical    advance
## 0.03333333 0.10000000 0.03333333 0.06666667 0.06666667 0.13333333 0.03333333
```

Above is the proportion of missing data for each variable. The univariate missingness rate is not very high for any variable - the maximum is 0.13. However, we are often more interested in multivariate missingness rates based on the variables in our analysis. For this example, we want to conduct a linear regression where the first variable "rating" is predicted by the other six variables. Therefore, we also want to see what the multivariate missingness rate is across all seven variables.

```r
miss_listwise <- sum(complete.cases(attitude_na)) / nrow(attitude_na)
print(miss_listwise)
```

```
## [1] 0.5333333
```

Here we see a much higher proportion of missing data - over half! If we were to use listwise deletion for the analysis, we would be using less than half the cases in the dataset. Lots of potential for bias and inefficiency in our results.

Impute the missing data

Let's go ahead and impute the missing data with the `mice` R package. We will see more of the `mice` R package later on in the blog post when we analyze and pool results from the multiply imputed data. For this example, we will create 10 imputed datasets.

```r
mice_obj <- mice(attitude_na, m = 10, method = "norm", printFlag = FALSE)
attitude_imp <- complete(mice_obj, action = "all", include = FALSE)
print(attitude_imp[1])
```

```
## $`1`
##      rating complaints privileges learning   raises critical  advance
## 1  58.04811   51.00000   30.00000 39.00000 61.00000 92.00000 45.00000
## 2  63.00000   64.00000   51.00000 54.00000 63.00000 73.00000 47.00000
## 3  71.00000   69.88867   68.00000 69.00000 76.00000 86.00000 48.00000
## 4  61.00000   63.00000   45.00000 47.00000 54.00000 84.00000 35.00000
## 5  81.00000   78.00000   56.00000 66.00000 71.42993 83.00000 47.00000
## 6  43.00000   55.00000   49.00000 44.00000 54.00000 49.00000 34.00000
## 7  58.00000   67.00000   42.00000 56.00000 66.00000 68.00000 35.00000
## 8  71.00000   75.00000   50.00000 55.00000 70.00000 66.00000 41.00000
## 9  72.00000   82.00000   72.00000 67.00000 71.00000 83.00000 31.00000
## 10 67.00000   61.00000   60.13021 47.00000 62.00000 80.00000 41.00000
## 11 64.00000   53.00000   53.00000 58.00000 58.00000 69.65986 34.00000
## 12 67.00000   71.51752   47.00000 39.00000 59.00000 74.00000 41.00000
## 13 69.00000   62.00000   57.00000 42.00000 55.00000 63.00000 35.20979
## 14 68.00000   69.12924   83.00000 45.00000 59.00000 77.00000 35.00000
## 15 77.00000   77.00000   54.00000 72.00000 79.00000 73.23265 46.00000
## 16 81.00000   90.00000   50.00000 72.00000 60.00000 54.00000 36.00000
## 17 74.00000   85.00000   64.00000 68.31819 79.00000 79.00000 63.00000
## 18 65.00000   60.00000   65.00000 75.00000 55.00000 80.00000 60.00000
## 19 65.00000   70.00000   46.00000 57.00000 75.00000 85.00000 46.00000
## 20 50.00000   58.00000   68.00000 54.00000 64.00000 78.00000 52.00000
## 21 50.00000   40.00000   33.00000 34.00000 43.00000 74.79197 33.00000
## 22 64.00000   61.00000   52.00000 62.00000 66.00000 80.00000 41.00000
## 23 53.00000   66.00000   52.00000 50.00000 63.00000 80.00000 37.00000
## 24 40.00000   37.00000   42.00000 58.00000 50.00000 57.00000 49.00000
## 25 63.00000   54.00000   42.00000 48.00000 66.00000 75.00000 33.00000
## 26 66.00000   77.00000   66.00000 63.00000 89.49276 76.00000 72.00000
## 27 78.00000   75.00000   58.00000 74.00000 80.00000 79.78280 49.00000
## 28 48.00000   57.00000   44.00000 45.00000 51.00000 83.00000 38.00000
## 29 85.00000   85.00000   71.00000 70.33588 77.00000 74.00000 55.00000
## 30 82.00000   82.00000   39.00000 59.00000 64.00000 78.00000 39.00000
```

The first of the 10 imputed datasets is printed above. It is the first element of a list with 10 elements containing 10 data.frames: one for every imputed dataset. The data.frames do not contain any missing values; instead, the missing values have been imputed and are the non-integer numbers. Each imputed dataset has different non-integer numbers for their imputed values. This is how multiple imputation models uncertainty due to missing data.

Now we will proceed to analyze and pool results from these multiply imputed data. I will showcase doing this 3 different ways in R: 1) manual calculations using the specific formulas, 2) the `mitools` R package, 3) the `mice` R package.

Manual Calculations

The manual calculations are taken from the book *Applied Missing Data Analysis* by Craig Enders (Enders, 2010). It is a book in the Methodology in the Social Sciences Series by Todd Little, a series of statistics books for social scientists that I highly recommend.

If you remember, our example analysis is a linear regression where the first variable "rating" is predicted by the other six variables in the data. Here is the regression formula in R:

```
frm <- v2frm(names(attitude))
print(frm)
```

```
## rating ~ complaints + privileges + learning + raises + critical +
##     advance
## <environment: 0x0000010e899457b0>
```

Let's go ahead and run this linear regression with each of the 10 imputed datasets resulting in 10 different regression models.

```
lm_imp <- lapply(X = attitude_imp, FUN = function(dat) lm(frm, data = dat))
lapply(X = lm_imp[1:3], FUN = summary) # summary.lm
```

```
## $`1`
##
## Call:
## lm(formula = frm, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -12.240  -3.302   1.292   4.288  10.953
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.057836  11.597088   0.350 0.729597
## complaints   0.661330   0.149324   4.429 0.000193 ***
## privileges  -0.030368   0.119633  -0.254 0.801871
## learning     0.214943   0.154806   1.388 0.178297
## raises      -0.005467   0.207409  -0.026 0.979198
## critical     0.175102   0.137367   1.275 0.215141
## advance     -0.143164   0.169335  -0.845 0.406571
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 6.634 on 23 degrees of freedom
## Multiple R-squared:  0.7381, Adjusted R-squared:  0.6698
## F-statistic:  10.8 on 6 and 23 DF,  p-value: 9.888e-06
##
##
## $`2`
##
## Call:
## lm(formula = frm, data = dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.5016  -3.2599   0.4988   4.4918  11.7119
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  8.30508    9.57475   0.867 0.394686
## complaints   0.66568    0.14246   4.673 0.000105 ***
## privileges  -0.14211    0.12572  -1.130 0.269975
## learning     0.14269    0.15405   0.926 0.363914
## raises       0.06215    0.22966   0.271 0.789093
## critical     0.13510    0.12200   1.107 0.279574
## advance     -0.05963    0.14795  -0.403 0.690615
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.601 on 23 degrees of freedom
## Multiple R-squared:  0.7385, Adjusted R-squared:  0.6703
## F-statistic: 10.83 on 6 and 23 DF,  p-value: 9.729e-06
##
##
## $`3`
##
## Call:
## lm(formula = frm, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -15.779  -5.441   2.056   4.240  12.614
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  4.222265  12.822818   0.329   0.7449
## complaints   0.643043   0.188622   3.409   0.0024 **
## privileges   0.094448   0.139005   0.679   0.5036
## learning     0.282931   0.169492   1.669   0.1086
## raises      -0.005408   0.240761  -0.022   0.9823
## critical     0.110778   0.127981   0.866   0.3957
## advance     -0.276319   0.179682  -1.538   0.1377
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.663 on 23 degrees of freedom
## Multiple R-squared:  0.7219, Adjusted R-squared:  0.6493
```

```
## F-statistic:  9.95 on 6 and 23 DF,  p-value: 1.899e-05
```

Here are the summaries from the first 3 of 10 regression models. As you can see, the coefficients, standard errors, and p-values are different for each model. This is because each analysis is using different numbers for the missing values.

As I mentioned in the beginning, **none of these summaries are our final results**. We want to *pool* the results from each of the 10 models together to obtain the final results that provide unbiased coefficients and appropriately sized standard errors. To do so, we need to extract statistical information from each regression model.

```
coef_imp <- lapply(X = lm_imp, FUN = coef)
print(coef_imp[1:3])
```

```
## $`1`
##  (Intercept)   complaints   privileges     learning       raises     critical
##  4.057835829  0.661329619 -0.030368173  0.214942544 -0.005467137  0.175102265
##      advance
## -0.143163989
##
## $`2`
## (Intercept)  complaints  privileges    learning      raises    critical
##  8.30507954  0.66567579 -0.14210822  0.14269179  0.06215128  0.13509853
##     advance
## -0.05963323
##
## $`3`
##  (Intercept)   complaints   privileges     learning       raises     critical
##  4.222265008  0.643042746  0.094448022  0.282930566 -0.005408141  0.110778386
##      advance
## -0.276319494
```

```
vcov_imp <- lapply(X = lm_imp, FUN = vcov) # vcov.lm
print(vcov_imp[1:3])
```

```
## $`1`
##              (Intercept)   complaints    privileges     learning        raises
## (Intercept) 134.49244566 -0.276665927 -1.557961e-01 -0.272229966 -2.820885e-02
## complaints   -0.27666593  0.022297653 -3.860095e-03 -0.007000684 -1.714124e-02
## privileges   -0.15579609 -0.003860095  1.431205e-02 -0.002849876 -1.997736e-05
## learning     -0.27222997 -0.007000684 -2.849876e-03  0.023964829 -6.362289e-03
## raises       -0.02820885 -0.017141239 -1.997736e-05 -0.006362289  4.301867e-02
## critical     -1.12714029  0.001663484 -6.952145e-04  0.003826410 -7.890817e-03
## advance      -0.10179418  0.008753517 -3.259637e-03 -0.007739305 -1.530715e-02
##                  critical      advance
## (Intercept) -1.1271402893 -0.101794181
## complaints   0.0016634842  0.008753517
## privileges  -0.0006952145 -0.003259637
## learning     0.0038264096 -0.007739305
## raises      -0.0078908173 -0.015307153
## critical     0.0188696257 -0.001600233
## advance     -0.0016002329  0.028674468
##
```

```
## $`2`
##             (Intercept)   complaints    privileges      learning       raises
## (Intercept) 91.67579649 -0.144901530 -0.1724851947  4.866995e-02 -0.453933709
## complaints  -0.14490153  0.020294448 -0.0066747319 -3.130182e-03 -0.017640340
## privileges  -0.17248519 -0.006674732  0.0158053680 -2.950208e-03  0.002257411
## learning     0.04866995 -0.003130182 -0.0029502077  2.373041e-02 -0.010893968
## raises      -0.45393371 -0.017640340  0.0022574109 -1.089397e-02  0.052742588
## critical    -0.42687855  0.003819786 -0.0001824772  2.699779e-05 -0.014507772
## advance     -0.28360063  0.003342721 -0.0042525908 -7.339344e-03 -0.002860102
##                 critical       advance
## (Intercept) -4.268785e-01 -0.2836006328
## complaints   3.819786e-03  0.0033427207
## privileges  -1.824772e-04 -0.0042525908
## learning     2.699779e-05 -0.0073393439
## raises      -1.450777e-02 -0.0028601020
## critical     1.488343e-02 -0.0008905623
## advance     -8.905623e-04  0.0218878610
##
## $`3`
##               (Intercept)   complaints    privileges      learning
## (Intercept) 164.424657979 -0.351598116 -0.3264455152 -0.1710983876
## complaints   -0.351598116  0.035578261 -0.0037165735 -0.0136947374
## privileges   -0.326445515 -0.003716574  0.0193223107 -0.0048205004
## learning     -0.171098388 -0.013694737 -0.0048205004  0.0287274208
## raises       -0.358434339 -0.028778148 -0.0020604732  0.0007337466
## critical     -1.149751866  0.003511584  0.0002981556  0.0014581340
## advance      -0.007114016  0.013101922 -0.0017150316 -0.0104729713
##                    raises      critical      advance
## (Intercept) -0.3584343386 -1.1497518657 -0.007114016
## complaints  -0.0287781480  0.0035115836  0.013101922
## privileges  -0.0020604732  0.0002981556 -0.001715032
## learning     0.0007337466  0.0014581340 -0.010472971
## raises       0.0579660371 -0.0060083363 -0.022509400
## critical    -0.0060083363  0.0163791653 -0.001251681
## advance     -0.0225093999 -0.0012516807  0.032285685
```

Above, we have extracted the regression coefficient and their sampling variance-covariance matrix (VCOV). I have also printed each piece of statistical information from the first three imputed datasets. While lists are great for storing complex R objects, they are not so great for mathematical operations. Here is where `str2str` comes in handy as its easier to do math on matrices and (3D+) arrays rather than lists or data.frames.

```
coef_mat <- lv2m(coef_imp, along = 1)
print(coef_mat[1:3, ])
```

```
##   (Intercept) complaints  privileges  learning       raises  critical
## 1    4.057836  0.6613296 -0.03036817 0.2149425 -0.005467137 0.1751023
## 2    8.305080  0.6656758 -0.14210822 0.1426918  0.062151280 0.1350985
## 3    4.222265  0.6430427  0.09444802 0.2829306 -0.005408141 0.1107784
##       advance
## 1 -0.14316399
## 2 -0.05963323
## 3 -0.27631949
```

```r
vcov_ary <- lm2a(vcov_imp)
print(vcov_ary[,, 1:3])
```

```
## , , 1
##
##              (Intercept)   complaints    privileges     learning        raises
## (Intercept) 134.49244566 -0.276665927 -1.557961e-01 -0.272229966 -2.820885e-02
## complaints   -0.27666593  0.022297653 -3.860095e-03 -0.007000684 -1.714124e-02
## privileges   -0.15579609 -0.003860095  1.431205e-02 -0.002849876 -1.997736e-05
## learning     -0.27222997 -0.007000684 -2.849876e-03  0.023964829 -6.362289e-03
## raises       -0.02820885 -0.017141239 -1.997736e-05 -0.006362289  4.301867e-02
## critical     -1.12714029  0.001663484 -6.952145e-04  0.003826410 -7.890817e-03
## advance      -0.10179418  0.008753517 -3.259637e-03 -0.007739305 -1.530715e-02
##                  critical      advance
## (Intercept) -1.1271402893 -0.101794181
## complaints   0.0016634842  0.008753517
## privileges  -0.0006952145 -0.003259637
## learning     0.0038264096 -0.007739305
## raises      -0.0078908173 -0.015307153
## critical     0.0188696257 -0.001600233
## advance     -0.0016002329  0.028674468
##
## , , 2
##
##              (Intercept)   complaints    privileges      learning       raises
## (Intercept) 91.67579649 -0.144901530 -0.1724851947  4.866995e-02 -0.453933709
## complaints  -0.14490153  0.020294448 -0.0066747319 -3.130182e-03 -0.017640340
## privileges  -0.17248519 -0.006674732  0.0158053680 -2.950208e-03  0.002257411
## learning     0.04866995 -0.003130182 -0.0029502077  2.373041e-02 -0.010893968
## raises      -0.45393371 -0.017640340  0.0022574109 -1.089397e-02  0.052742588
## critical    -0.42687855  0.003819786 -0.0001824772  2.699779e-05 -0.014507772
## advance     -0.28360063  0.003342721 -0.0042525908 -7.339344e-03 -0.002860102
##                  critical       advance
## (Intercept) -4.268785e-01 -0.2836006328
## complaints   3.819786e-03  0.0033427207
## privileges  -1.824772e-04 -0.0042525908
## learning     2.699779e-05 -0.0073393439
## raises      -1.450777e-02 -0.0028601020
## critical     1.488343e-02 -0.0008905623
## advance     -8.905623e-04  0.0218878610
##
## , , 3
##
##               (Intercept)   complaints    privileges      learning
## (Intercept) 164.424657979 -0.351598116 -0.3264455152 -0.1710983876
## complaints   -0.351598116  0.035578261 -0.0037165735 -0.0136947374
## privileges   -0.326445515 -0.003716574  0.0193223107 -0.0048205004
## learning     -0.171098388 -0.013694737 -0.0048205004  0.0287274208
## raises       -0.358434339 -0.028778148 -0.0020604732  0.0007337466
## critical     -1.149751866  0.003511584  0.0002981556  0.0014581340
## advance      -0.007114016  0.013101922 -0.0017150316 -0.0104729713
##                    raises      critical      advance
## (Intercept) -0.3584343386 -1.1497518657 -0.007114016
```

```
## complaints  -0.0287781480  0.0035115836  0.013101922
## privileges  -0.0020604732  0.0002981556 -0.001715032
## learning     0.0007337466  0.0014581340 -0.010472971
## raises       0.0579660371 -0.0060083363 -0.022509400
## critical    -0.0060083363  0.0163791653 -0.001251681
## advance     -0.0225093999 -0.0012516807  0.032285685
```

As you can see, we have our regression coefficients in a matrix and their sampling variance-covariance matrices (VCOV) in a 3D array. Now we can easily apply the pooling formulas.

Regression coefficients

First let's get the regression coefficients by taking the mean across each imputed dataset. The formula is the usual mean we calculate in statistics:

$$b_{pool} = \frac{\sum_{i=1}^{m} b_i}{m}$$

```
coef_pool <- colMeans(coef_mat)
print(coef_pool)
```

```
## (Intercept)  complaints  privileges    learning      raises    critical
##  8.25122733  0.63116347 -0.05715522  0.17977462  0.05618392  0.14488926
##     advance
## -0.16062333
```

Sampling VCOV

The formula for the sampling variance-covariance matrix (VCOV) is a little more complicated. We need to create separate matrices that quantity the uncertainty due to sample size (i.e., within imputed datasets), the uncertainty due to missing data (i.e., between imputed datasets), and the uncertainty due to finite simulation of the missing data (i.e., using 10 vs. `Inf` imputed datasets). Adding those up will give us the total sampling error.

Let's start first with the uncertainty due to sample size. The formula is simply the average of each VCOV element across the m imputed datasets:

$$VCOV_{sample.size} = \frac{\sum_{i=1}^{m} VCOV_i}{m}$$

```
sample_size <- apply(X = vcov_ary, MARGIN = c(1,2), FUN = mean)
print(sample_size)
```

```
##              (Intercept)   complaints    privileges     learning        raises
## (Intercept) 129.78697119 -0.213153877 -0.2264302331 -0.022016051 -0.4263992691
## complaints   -0.21315388  0.027653619 -0.0050606333 -0.009956635 -0.0208017795
## privileges   -0.22643023 -0.005060633  0.0168729633 -0.003500966  0.0007693454
## learning     -0.02201605 -0.009956635 -0.0035009656  0.028648535 -0.0078686288
## raises       -0.42639927 -0.020801780  0.0007693454 -0.007868629  0.0544095862
## critical     -0.88543966  0.001982685 -0.0007975125  0.003342900 -0.0106501547
## advance      -0.14599702  0.009221505 -0.0029853261 -0.011680552 -0.0111179123
##                  critical      advance
## (Intercept) -0.8854396553 -0.145997020
```

```
## complaints   0.0019826855  0.009221505
## privileges  -0.0007975125 -0.002985326
## learning     0.0033428999 -0.011680552
## raises      -0.0106501547 -0.011117912
## critical     0.0184171214 -0.002337021
## advance     -0.0023370214  0.029054300
```

Next is the uncertainty due to missing data. Here we quantify how much the regression coefficients from each imputed dataset differ from the pooled value. The formula is the same as for a covariance matrix of data:

$$VCOV_{missing.data} = \frac{\sum_{i=1}^{m}(b_i - b_{pool})^T(b_i - b_{pool})}{m-1}$$

```
coef_dev <- coef_mat - matrix(coef_pool, byrow = TRUE, nrow = nrow(coef_mat),
   ncol = ncol(coef_mat))
coef_dev2 <- t(coef_dev) %*% coef_dev
dimnames(coef_dev2) <- dimnames(sample_size)
m <- nlay(vcov_ary)
missing_data <- coef_dev2 / (m - 1)
print(missing_data)
```

```
##              (Intercept)   complaints   privileges     learning       raises
## (Intercept) 20.662380773 -0.020074930 -0.001024636 -0.062879268  0.034978847
## complaints  -0.020074930  0.002228301  0.002173368  0.000571672 -0.003620090
## privileges  -0.001024636  0.002173368  0.008205213  0.003720500 -0.004266201
## learning    -0.062879268  0.000571672  0.003720500  0.003734721 -0.001304057
## raises       0.034978847 -0.003620090 -0.004266201 -0.001304057  0.007060274
## critical    -0.286250100 -0.001383792 -0.005277430 -0.002500645  0.001833104
## advance      0.082142510  0.001144886 -0.003409559 -0.003166153 -0.001525757
##                 critical      advance
## (Intercept) -0.286250100  0.082142510
## complaints  -0.001383792  0.001144886
## privileges  -0.005277430 -0.003409559
## learning    -0.002500645 -0.003166153
## raises       0.001833104 -0.001525757
## critical     0.008733128  0.001232488
## advance      0.001232488  0.005116761
```

Last we have the uncertainty due to finite simulation. Here we are penalizing ourselves for quantifying the uncertainty due to missing data with a finite number of imputed datasets. In an ideal world, we would use an infinite number of imputed datasets; however, that is not possible (I return to the issue of how many imputed datasets to create later on in the blog post). The formula is a simple function of the uncertainty due to missing data and the number of imputed datasets:

$$VCOV_{finite.simulation} = \frac{VCOV_{missing.data}}{m}$$

```
finite_simulation <- missing_data / m
print(finite_simulation)
```

```
##               (Intercept)    complaints    privileges      learning
## (Intercept)  2.0662380773 -0.0020074930 -0.0001024636 -0.0062879268
## complaints  -0.0020074930  0.0002228301  0.0002173368  0.0000571672
```

```
## privileges  -0.0001024636  0.0002173368  0.0008205213  0.0003720500
## learning    -0.0062879268  0.0000571672  0.0003720500  0.0003734721
## raises       0.0034978847 -0.0003620090 -0.0004266201 -0.0001304057
## critical    -0.0286250100 -0.0001383792 -0.0005277430 -0.0002500645
## advance      0.0082142510  0.0001144886 -0.0003409559 -0.0003166153
##                    raises      critical       advance
## (Intercept)  0.0034978847 -0.0286250100  0.0082142510
## complaints  -0.0003620090 -0.0001383792  0.0001144886
## privileges  -0.0004266201 -0.0005277430 -0.0003409559
## learning    -0.0001304057 -0.0002500645 -0.0003166153
## raises       0.0007060274  0.0001833104 -0.0001525757
## critical     0.0001833104  0.0008733128  0.0001232488
## advance     -0.0001525757  0.0001232488  0.0005116761
```

Now we can add up all the sampling variance-covariance matrix components into the total sampling error:

$$VCOV_{total.error} = VCOV_{sample.size} + VCOV_{missing.data} + VCOV_{missing.data}$$

```
total_error <- sample_size + missing_data + finite_simulation
print(total_error)
```

```
##              (Intercept)    complaints    privileges      learning       raises
## (Intercept) 152.51559004 -0.2352362999 -0.2275573328 -0.0911832455 -0.387922537
## complaints   -0.23523630  0.0301047502 -0.0026699284 -0.0093277960 -0.024783878
## privileges   -0.22755733 -0.0026699284  0.0258986977  0.0005915847 -0.003923476
## learning     -0.09118325 -0.0093277960  0.0005915847  0.0327567279 -0.009303091
## raises       -0.38792254 -0.0247838782 -0.0039234756 -0.0093030915  0.062175888
## critical     -1.20031477  0.0004605148 -0.0066026853  0.0005921905 -0.008633740
## advance      -0.05564026  0.0104808796 -0.0067358407 -0.0151633197 -0.012796245
##                  critical       advance
## (Intercept) -1.2003147656 -0.0556402586
## complaints   0.0004605148  0.0104808796
## privileges  -0.0066026853 -0.0067358407
## learning     0.0005921905 -0.0151633197
## raises      -0.0086337399 -0.0127962448
## critical     0.0280235618 -0.0009812848
## advance     -0.0009812848  0.0346827369
```

FMI and df

The next step is computing the fraction of missing information (FMI) to quantify what proportion of the total sampling error is due to missing data (and finite simulation). This will also allow us to then calculate appropriate degrees of freedom (df) for each regression coefficient. The formula for FMI is fairly simple, which makes it easy to understand how exactly to interpret the statistic. Note, that the FMI is not necessarily the same as the proportion of missing data and often depends on the particular analysis just as much as the amount of missing data. Also keep in mind that because of the uncertainty due to finite simulation (i.e., using 10 vs. `Inf` imputed datasets), the FMI will be non-zero even when there is no missing data for the variables in a particular analysis. The formula only uses the sampling variances (SE^2; squared standard errors) and not the sampling covariances:

$$FMI = \frac{SE^2_{missing.data} + SE^2_{finite.simulation}}{SE^2_{total.error}}$$

```r
fmi <- (diag(missing_data) + diag(finite_simulation)) / diag(total_error)
print(fmi)
```

```
## (Intercept)  complaints  privileges    learning      raises    critical
##  0.14902489  0.08142007  0.34850148  0.12541525  0.12490858  0.34279869
##     advance
##  0.16228352
```

Now, we will use the FMI to compute the appropriate degrees of freedom (df). The df calculations are a bit more complicated than the FMI ones and I find it harder to understand exactly what it going on from just looking at the formula. A good summary is that the sample size degrees of freedom (aka "complete" degrees of freedom) are penalized based on the FMI. Sometimes this is called the "small-sample" degrees of freedom formula for multiply imputed data. Of course, another option is to use z-tests without degrees of freedom. Note, this is an exception where we don't need to compute the complete degrees of freedom separately for each imputed dataset since the complete degrees of freedom are all the same from having the same sample size.

$$df_{imputation} = \frac{1}{\frac{1}{\frac{m-1}{FMI^2}} + \frac{1}{(1-FMI)\left(\frac{df_{complete}+1}{df_{complete}+3}\right)df_{complete}}}$$

```r
df_com <- df.residual(lm_imp[[1]])
v <- (m - 1) / fmi^2
vhat <- (1 - fmi) * ((df_com + 1) / (df_com + 3)) * df_com
df_imp <- (v^(-1) + vhat^(-1))^(-1)
print(df_imp)
```

```
## (Intercept)  complaints  privileges    learning      raises    critical
##    17.29578    19.22598    11.65611    17.98450    17.99915    11.80268
##     advance
##    16.90554
```

Final results

Now we are ready to compute our t-values and p-values for hypothesis testing.

```r
se_pool <- sqrt(diag(total_error))
t_pool <- coef_pool / se_pool
p_pool <- 2 * pt(q = abs(t_pool), df = df_imp, lower.tail = F)
```

As well as our confidence intervals for parameter estimation.

```r
crit_pool <- qt(p = 1 - (.05 / 2), df = df_imp, lower.tail = T)
hw_pool <- se_pool * crit_pool
ci_pool <- cbind("lwr" = coef_pool - hw_pool, "upr" = coef_pool + hw_pool)
```

Alright, let's put all the results together in a data.frame to make them easy to view.

```r
result_pool <- data.frame("est" = coef_pool, "se" = se_pool, "t" = t_pool,
   "df" = df_imp, "p" = p_pool, ci_pool, "fmi" = fmi)
print(result_pool)
```

```
##                    est         se          t       df          p         lwr
## (Intercept)  8.25122733 12.3497202  0.6681307 17.29578 0.512868414 -17.7705033
## complaints   0.63116347  0.1735072  3.6376787 19.22598 0.001724592   0.2682974
## privileges  -0.05715522  0.1609307 -0.3551542 11.65611 0.728817486  -0.4089439
## learning     0.17977462  0.1809882  0.9932947 17.98450 0.333743577  -0.2004910
## raises       0.05618392  0.2493509  0.2253207 17.99915 0.824268278  -0.4676847
## critical     0.14488926  0.1674024  0.8655149 11.80268 0.404006318  -0.2205267
## advance     -0.16062333  0.1862330 -0.8624858 16.90554 0.400490198  -0.5537080
##                    upr        fmi
## (Intercept) 34.2729579 0.14902489
## complaints   0.9940295 0.08142007
## privileges   0.2946335 0.34850148
## learning     0.5600402 0.12541525
## raises       0.5800526 0.12490858
## critical     0.5103052 0.34279869
## advance      0.2324613 0.16228352
```

Adjusted FMI and RIV

I want to mention an adjusted FMI that is often reported instead of the simpler FMI. It provides an additional adjustment for finite simulation (i.e., using 10 imputed datasets instead of `Inf`) that makes the FMI estimate unbiased. The adjustment uses apart of the degree of freedom formula we just calculated. As the number of imputed datasets increases, the unadjusted FMI asymptotically approaches the adjusted FMI.

$$FMI_{adjusted} = FMI + \frac{\frac{2*SE^2_{sample.size}}{\frac{m-1}{FMI^2}+3}}{SE^2_{total.error}}$$

.

```
fmi_adj <- fmi + (((2 * diag(sample_size)) / (v + 3)) / diag(total_error))
print(fmi_adj)
```

```
## (Intercept)  complaints  privileges    learning      raises    critical
##   0.1531938   0.0827703   0.3654010   0.1284563   0.1279269   0.3593137
##     advance
##   0.1671435
```

Another useful piece of statistical information for quantifying the influence of missing data on total sampling error is the relative increase in variance (RIV), which is the ratio of the sampling error due to missing data (and finite simulation) vs. sample size. When the FMI = 0.5 - meaning that half of the total sampling error is due to missing data (and finite simulation), the RIV = 1.0 - meaning the sampling variance increases by 100% (aka doubles) when going from the sampling variance due to sample size to the total sampling error. The formula is:

$$RIV = \frac{FMI}{1 - FMI}$$

.

```
riv <- fmi / (1 - fmi)
print(riv)
```

```
## (Intercept)  complaints  privileges    learning      raises    critical
##  0.17512250  0.08863689  0.53492289  0.14339976  0.14273775  0.52160379
##     advance
##  0.19372130
```

We can also calculate an adjusted version of the RIV from the adjusted FMI that makes the estimate unbiased.

```
riv_adj <- fmi_adj / (1 - fmi_adj)
print(riv_adj)
```

```
## (Intercept)  complaints  privileges    learning      raises    critical
##  0.18090769  0.09023945  0.57579827  0.14738935  0.14669293  0.56082614
##     advance
##  0.20068709
```

Again, let's put all the results together in a data.frame to make them easy to view.

```
cbind(result_pool) <- data.frame("fmi_adj" = fmi_adj, "riv" = riv, "riv_adj" = riv_adj)
print(result_pool)
```

```
##                    est         se          t       df           p         lwr
## (Intercept)  8.25122733 12.3497202  0.6681307 17.29578 0.512868414 -17.7705033
## complaints   0.63116347  0.1735072  3.6376787 19.22598 0.001724592   0.2682974
## privileges  -0.05715522  0.1609307 -0.3551542 11.65611 0.728817486  -0.4089439
## learning     0.17977462  0.1809882  0.9932947 17.98450 0.333743577  -0.2004910
## raises       0.05618392  0.2493509  0.2253207 17.99915 0.824268278  -0.4676847
## critical     0.14488926  0.1674024  0.8655149 11.80268 0.404006318  -0.2205267
## advance     -0.16062333  0.1862330 -0.8624858 16.90554 0.400490198  -0.5537080
##                    upr        fmi   fmi_adj        riv    riv_adj
## (Intercept) 34.2729579 0.14902489 0.1531938 0.17512250 0.18090769
## complaints   0.9940295 0.08142007 0.0827703 0.08863689 0.09023945
## privileges   0.2946335 0.34850148 0.3654010 0.53492289 0.57579827
## learning     0.5600402 0.12541525 0.1284563 0.14339976 0.14738935
## raises       0.5800526 0.12490858 0.1279269 0.14273775 0.14669293
## critical     0.5103052 0.34279869 0.3593137 0.52160379 0.56082614
## advance      0.2324613 0.16228352 0.1671435 0.19372130 0.20068709
```

That concludes our venture into the specific formulas with manual calculations. Now that was fun to do (and hopefully educational), but it was also a lot of work - knowledge of both mathematical formulas and R coding was needed. Of course one of the best parts about R is there is a (open source) package for just about everything. Let's look at what package options there are for analyzing and pooling results from multiply imputed data.

mitools R package

When using the mitools R package for analyzing and pooling results from multiply imputed data, the first step is converting the list of imputed datasets into an object of class `impuationList` that the `mitools` package can work with.

```
library(mitools)
attitude_impList <- imputationList(attitude_imp)
summary(attitude_impList)
```

```
##             Length Class  Mode
## imputations 10     mild   list
## call         2     -none- call
```

As you can see, the `imputationList` object is simply a list with two elements: 1) list of the imputed datasets, and 2) the call. The `mitools` package has a convenient `update` method for `imputationList` objects. For example, if we want to create a new column in the data that is a total score of the predictors, we can easily do that with non-standard evaluation (here is a YouTube that I think does a good job explaining standard vs. non-standard evalution in R).

```
attitude_impList <- update(attitude_impList, total = (complaints + privileges +
      learning + raises + critical + advance) / 6) # update.imputationList
lapply(X = attitude_impList[["imputations"]][1:3], FUN = head)
```

```
## $`1`
##     rating complaints privileges learning   raises critical advance    total
## 1 58.04811   51.00000         30       39 61.00000       92      45 53.00000
## 2 63.00000   64.00000         51       54 63.00000       73      47 58.66667
## 3 71.00000   69.88867         68       69 76.00000       86      48 69.48145
## 4 61.00000   63.00000         45       47 54.00000       84      35 54.66667
## 5 81.00000   78.00000         56       66 71.42993       83      47 66.90499
## 6 43.00000   55.00000         49       44 54.00000       49      34 47.50000
##
## $`2`
##     rating complaints privileges learning   raises critical advance    total
## 1 60.81278   51.00000         30       39 61.00000       92      45 53.00000
## 2 63.00000   64.00000         51       54 63.00000       73      47 58.66667
## 3 71.00000   76.42267         68       69 76.00000       86      48 70.57045
## 4 61.00000   63.00000         45       47 54.00000       84      35 54.66667
## 5 81.00000   78.00000         56       66 72.65548       83      47 67.10925
## 6 43.00000   55.00000         49       44 54.00000       49      34 47.50000
##
## $`3`
##     rating complaints privileges learning   raises critical advance    total
## 1 32.53339   51.00000         30       39 61.00000       92      45 53.00000
## 2 63.00000   64.00000         51       54 63.00000       73      47 58.66667
## 3 71.00000   82.63919         68       69 76.00000       86      48 71.60653
## 4 61.00000   63.00000         45       47 54.00000       84      35 54.66667
## 5 81.00000   78.00000         56       66 73.11145       83      47 67.18524
## 6 43.00000   55.00000         49       44 54.00000       49      34 47.50000
```

You can see that the `imputationList` method for `update` added a "total" column in each imputed dataset. Now, to do analyses, we use the `imputationList` method for `with`, which again uses non-standard evaluation. Let's run our same linear regression model for each imputed dataset.

```
lm_imp2 <- with(attitude_impList, lm(rating ~ complaints + privileges +
      learning + raises + critical + advance))
head(lm_imp2, n = 3)
```

```
## $`1`
##
## Call:
## lm(formula = rating ~ complaints + privileges + learning + raises +
##     critical + advance)
##
## Coefficients:
## (Intercept)   complaints   privileges     learning       raises     critical
```

```
##    4.057836     0.661330    -0.030368     0.214943    -0.005467     0.175102
##     advance
##   -0.143164
##
##
## $`2`
##
## Call:
## lm(formula = rating ~ complaints + privileges + learning + raises +
##     critical + advance)
##
## Coefficients:
## (Intercept)   complaints   privileges     learning       raises     critical
##     8.30508      0.66568     -0.14211      0.14269      0.06215      0.13510
##     advance
##    -0.05963
##
##
## $`3`
##
## Call:
## lm(formula = rating ~ complaints + privileges + learning + raises +
##     critical + advance)
##
## Coefficients:
## (Intercept)   complaints   privileges     learning       raises     critical
##    4.222265     0.643043     0.094448     0.282931    -0.005408     0.110778
##     advance
##   -0.276319
```

Again, we find ourselves with a list of models - one for each imputed dataset.

MIextract

The next step is to extract the statistical information of interest. To do this, we use the `MIextract` function, which is basically just a wrapper for `lapply(X = results, FUN = fun)`.

```
coef_imp2 <- MIextract(results = lm_imp2, fun = coef)
head(coef_imp2, n = 3)
```

```
## $`1`
##  (Intercept)   complaints   privileges     learning       raises     critical
##  4.057835829  0.661329619 -0.030368173  0.214942544 -0.005467137  0.175102265
##      advance
## -0.143163989
##
## $`2`
## (Intercept)  complaints  privileges    learning      raises    critical
##  8.30507954  0.66567579 -0.14210822  0.14269179  0.06215128  0.13509853
##     advance
## -0.05963323
##
## $`3`
```

```
##  (Intercept)   complaints   privileges     learning       raises     critical
##  4.222265008  0.643042746  0.094448022  0.282930566 -0.005408141  0.110778386
##      advance
## -0.276319494
```

```
vcov_imp2 <- MIextract(results = lm_imp2, fun = vcov) # vcov.lm
head(vcov_imp2, n = 3)
```

```
## $`1`
##              (Intercept)   complaints    privileges     learning        raises
## (Intercept) 134.49244566 -0.276665927 -1.557961e-01 -0.272229966 -2.820885e-02
## complaints   -0.27666593  0.022297653 -3.860095e-03 -0.007000684 -1.714124e-02
## privileges   -0.15579609 -0.003860095  1.431205e-02 -0.002849876 -1.997736e-05
## learning     -0.27222997 -0.007000684 -2.849876e-03  0.023964829 -6.362289e-03
## raises       -0.02820885 -0.017141239 -1.997736e-05 -0.006362289  4.301867e-02
## critical     -1.12714029  0.001663484 -6.952145e-04  0.003826410 -7.890817e-03
## advance      -0.10179418  0.008753517 -3.259637e-03 -0.007739305 -1.530715e-02
##                  critical      advance
## (Intercept) -1.1271402893 -0.101794181
## complaints   0.0016634842  0.008753517
## privileges  -0.0006952145 -0.003259637
## learning     0.0038264096 -0.007739305
## raises      -0.0078908173 -0.015307153
## critical     0.0188696257 -0.001600233
## advance     -0.0016002329  0.028674468
##
## $`2`
##              (Intercept)   complaints    privileges      learning       raises
## (Intercept) 91.67579649 -0.144901530 -0.1724851947  4.866995e-02 -0.453933709
## complaints  -0.14490153  0.020294448 -0.0066747319 -3.130182e-03 -0.017640340
## privileges  -0.17248519 -0.006674732  0.0158053680 -2.950208e-03  0.002257411
## learning     0.04866995 -0.003130182 -0.0029502077  2.373041e-02 -0.010893968
## raises      -0.45393371 -0.017640340  0.0022574109 -1.089397e-02  0.052742588
## critical    -0.42687855  0.003819786 -0.0001824772  2.699779e-05 -0.014507772
## advance     -0.28360063  0.003342721 -0.0042525908 -7.339344e-03 -0.002860102
##                  critical       advance
## (Intercept) -4.268785e-01 -0.2836006328
## complaints   3.819786e-03  0.0033427207
## privileges  -1.824772e-04 -0.0042525908
## learning     2.699779e-05 -0.0073393439
## raises      -1.450777e-02 -0.0028601020
## critical     1.488343e-02 -0.0008905623
## advance     -8.905623e-04  0.0218878610
##
## $`3`
##               (Intercept)   complaints    privileges      learning
## (Intercept) 164.424657979 -0.351598116 -0.3264455152 -0.1710983876
## complaints   -0.351598116  0.035578261 -0.0037165735 -0.0136947374
## privileges   -0.326445515 -0.003716574  0.0193223107 -0.0048205004
## learning     -0.171098388 -0.013694737 -0.0048205004  0.0287274208
## raises       -0.358434339 -0.028778148 -0.0020604732  0.0007337466
## critical     -1.149751866  0.003511584  0.0002981556  0.0014581340
## advance      -0.007114016  0.013101922 -0.0017150316 -0.0104729713
##                    raises      critical      advance
```

```
## (Intercept) -0.3584343386 -1.1497518657 -0.007114016
## complaints  -0.0287781480  0.0035115836  0.013101922
## privileges  -0.0020604732  0.0002981556 -0.001715032
## learning     0.0007337466  0.0014581340 -0.010472971
## raises       0.0579660371 -0.0060083363 -0.022509400
## critical    -0.0060083363  0.0163791653 -0.001251681
## advance     -0.0225093999 -0.0012516807  0.032285685
```

MIcombine

The last step is to pool the statistial information together into our final results. This is where the `mitools` packge really saves us time with the `MIcombine` function.

```
df_com2 <- df.residual(lm_imp2[[1]])
all_pool <- MIcombine(results = coef_imp2, variances = vcov_imp2, df.complete = df_com2)
coef_pool2 <- all_pool[["coefficients"]]
se_pool2 <- sqrt(diag(all_pool[["variance"]]))
df_imp2 <- all_pool[["df"]]
t_pool2 <- coef_pool2 / se_pool2
p_pool2 <- 2 * pt(q = abs(t_pool2), df = df_imp2, lower.tail = F)
fmi2 <- all_pool[["missinfo"]]
result_pool2 <- data.frame("est" = coef_pool2, "se" = se_pool2, "t" = t_pool2,
   "df" = df_imp2, "p" = p_pool2, "fmi" = fmi2)
print(result_pool2)
```

```
##                    est         se          t       df           p       fmi
## (Intercept)  8.25122733 12.3497202  0.6681307 17.52304 0.512755330 0.2319536
## complaints   0.63116347  0.1735072  3.6376787 19.36730 0.001708144 0.1635560
## privileges  -0.05715522  0.1609307 -0.3551542 11.97585 0.728650431 0.4355080
## learning     0.17977462  0.1809882  0.9932947 18.18532 0.333600117 0.2079804
## raises       0.05618392  0.2493509  0.2253207 18.19936 0.824240578 0.2074669
## critical     0.14488926  0.1674024  0.8655149 12.12224 0.403560564 0.4297172
## advance     -0.16062333  0.1862330 -0.8624858 17.14598 0.400323252 0.2454482
```

Comparison to manual calculations

Let's see if our manual calculations gave us the same results as the `mitools` R package.

```
all.equal(coef_pool, coef_pool2)
```

```
## [1] TRUE
```

```
all.equal(se_pool, se_pool2)
```

```
## [1] TRUE
```

```
all.equal(t_pool, t_pool2)
```

```
## [1] TRUE
```

```r
all.equal(df_imp, df_imp2)
```

```
## [1] "Mean relative difference: 0.01461291"
```

```r
all.equal(p_pool, p_pool2)
```

```
## [1] "Mean relative difference: 0.0003370168"
```

```r
all.equal(fmi, fmi2)
```

```
## [1] "Mean relative difference: 0.440122"
```

The regression coefficients and standard errors are the same, but the degrees of freedom (df), p-values, and fraction of missing information (FMI) are different! This is because `mitools` calculates the df differently than we did in the manual calculations from Enders, 2010. This is an example of where I got confused when first learning about pooling with multiple imputation. Different books, R packages, statisticians, etc. seem to use slightly different formulas for the FMI and df. Instead of using the unadjusted FMI in the df formula, `mitools` uses the unadjusted relative increase in variance (RIV). An equivalent formula to the one above is...

$$RIV = \frac{SE^2_{missing.data} + SE^2_{finite.simulation}}{SE^2_{sample.size}}$$

.

If we replace the FMI with the RIV in our manual calculations, then we can reproduce the `mitools` df values:

```r
riv <- (diag(missing_data) + diag(finite_simulation)) / diag(sample_size)
v1 <- (m - 1) * (1 + (1 / riv))^2
vhat1 <- (diag(sample_size) / (diag(sample_size) + diag(missing_data))) *
   ((df_com + 1) / (df_com + 3)) * df_com
df_imp1 <- (v1^(-1) + vhat1^(-1))^(-1)
all.equal(df_imp1, df_imp2)
```

```
## [1] TRUE
```

With the same df, we now get the same p-values:

```r
p_pool1 <- 2 * pt(q = abs(t_pool), df = df_imp1, lower.tail = F)
all.equal(p_pool1, p_pool2)
```

```
## [1] TRUE
```

Regarding the FMI, `mitools` provides the adjusted FMI, but does a different adjustment than Enders, 2010. In addition to a correction for using finite simulation (i.e., 10 vs. `Inf` imputed dataets), `mitools` adds a correction for the sample size by incorperating the df:

$$FMI_{adjusted} = FMI + \frac{\frac{2*SE^2_{sample.size}}{df_{imputation}+3}}{SE^2_{total.error}}$$

(An equivalent formula for it that uses the RIV is... $FMI_{adjusted} = \frac{\left(\frac{RIV+2}{df_{imputation}+3}\right)}{RIV+1}$).

Once we add in that correction, we get the same FMI.

```
fmi_adj1 <- fmi + (((2 * diag(sample_size)) / (df_imp1 + 3)) / diag(total_error))
all.equal(fmi_adj1, fmi2)
```

```
## [1] TRUE
```

Finally, let's add in our confidence interval to the data.frame of `mitools` results. Of course, the confidence intervals will also be slightly different than from the manual calculations due to the different df; however, with the same df, the confidence intervals would be the same as well.

```
crit_pool2 <- qt(p = 1 - (.05 / 2), df = df_imp, lower.tail = T)
hw_pool2 <- se_pool2 * crit_pool2
ci_pool2 <- cbind("lwr" = coef_pool2 - hw_pool2, "upr" = coef_pool2 + hw_pool2)
cbind(result_pool2, after = "p") <- m2d(ci_pool2)
names(result_pool2)[names(result_pool2) == "data_after"] <- "fmi" # this is a str2str bug
print(result_pool2)
```

```
##                     est         se          t       df           p         lwr
## (Intercept)  8.25122733 12.3497202  0.6681307 17.52304 0.512755330 -17.7705033
## complaints   0.63116347  0.1735072  3.6376787 19.36730 0.001708144   0.2682974
## privileges  -0.05715522  0.1609307 -0.3551542 11.97585 0.728650431  -0.4089439
## learning     0.17977462  0.1809882  0.9932947 18.18532 0.333600117  -0.2004910
## raises       0.05618392  0.2493509  0.2253207 18.19936 0.824240578  -0.4676847
## critical     0.14488926  0.1674024  0.8655149 12.12224 0.403560564  -0.2205267
## advance     -0.16062333  0.1862330 -0.8624858 17.14598 0.400323252  -0.5537080
##                     upr       fmi
## (Intercept) 34.2729579 0.2319536
## complaints   0.9940295 0.1635560
## privileges   0.2946335 0.4355080
## learning     0.5600402 0.2079804
## raises       0.5800526 0.2074669
## critical     0.5103052 0.4297172
## advance      0.2324613 0.2454482
```

The fact that our manual calculations based on Enders, 2010 and the `mitools` functions gave us slightly different results is somewhat dissatisfying. Practically, the final results were essentially the same, providing the same p-values and confidence intervals to the third decimal place. But which degrees of freedom is correct and which fraction of missing information is correct? We do have another R package that might be able to provide a third opinion for us: `mice`. The same package that created the 10 imputed datasets has functions for analyzing and pooling results from multiply imputed data.

mice R package

Let's start by converting the list of imputed datasets to an object that the `mice` package can work with: `mids` S3 object. To do so, we need to append the original data with missing values.

```
library(mice)
tmp <- append(attitude_imp, values = list("0" = attitude_na), after = 0)
attitude_long <- ld2d(tmp, rtn.listnames.nm = ".imp", rtn.rownames.nm = ".id")
attitude_mids <- as.mids(attitude_long)
summary.default(attitude_mids)
```

```
##                 Length Class           Mode
## data              7    data.frame      list
## imp               7    -none-          list
## m                 1    -none-          numeric
## where           210    -none-          logical
## blocks            7    -none-          list
## call              7    -none-          call
## nmis              7    -none-          numeric
## method            7    -none-          character
## predictorMatrix  49    -none-          numeric
## visitSequence     7    -none-          character
## formulas          7    -none-          list
## post              7    -none-          character
## blots             7    -none-          list
## ignore           30    -none-          logical
## seed              1    -none-          logical
## iteration         1    -none-          numeric
## lastSeedValue   626    -none-          numeric
## chainMean         0    -none-          logical
## chainVar          0    -none-          logical
## loggedEvents      0    -none-          NULL
## version           1    package_version list
## date              1    Date            numeric
```

A mids object is a list with 22 elements. Most are for when you impute missing data with the `mice` function. Since we are starting with already imputed data, most elements are meaningless or empty. The imputed values are in the second "imp" element as a list with an element for each variable.

```
print(attitude_mids[["imp"]])
```

```
## $rating
##          1        2        3        4        5        6        7        8
## 1 58.04811 60.81278 32.53339 69.46543 74.91704 62.52202 57.86417 36.27887
##          9      10
## 1 69.64834 61.1879
##
## $complaints
##           1        2        3        4        5        6        7        8
## 3  69.88867 76.42267 82.63919 86.09762 61.81434 77.34001 73.77120 77.22803
## 12 71.51752 75.16151 59.31367 63.06432 66.13770 50.49449 66.63144 64.32393
## 14 69.12924 78.60887 62.85754 66.59771 77.08061 78.74188 59.14375 74.97034
##           9       10
## 3  83.70844 73.73677
## 12 66.00595 69.97001
## 14 70.87920 60.81478
##
## $privileges
##           1        2        3        4        5        6        7        8
## 10 60.13021 45.54125 48.71262 56.92266 43.85634 46.84978 44.34241 63.81759
##           9       10
## 10 72.20713 52.70609
##
## $learning
```

```
##           1        2        3        4        5        6        7        8
## 17 68.31819 67.87423 56.56904 85.46594 93.87689 84.23394 73.55026 75.70348
## 29 70.33588 61.91956 87.24336 82.74795 85.26030 71.60821 76.42411 79.15639
##           9       10
## 17 67.70135 77.82672
## 29 81.82931 69.30371
##
## $raises
##           1        2        3        4        5        6        7        8
## 5  71.42993 72.65548 73.11145 72.32616 84.97552 76.30432 64.40118 66.49822
## 26 89.49276 62.77665 91.92557 72.91777 81.30230 85.12116 78.07274 83.71178
##           9       10
## 5  62.40609 65.88702
## 26 70.16843 63.13036
##
## $critical
##           1         2         3        4        5        6         7        8
## 11 69.65986  78.37677  88.20755 82.16038 66.73054 74.06861 107.75646 67.35224
## 15 73.23265  97.39982  68.02526 82.07466 80.97156 81.99870  80.21335 69.27362
## 21 74.79197  60.25362  96.77726 52.53159 86.55375 66.19616  81.96623 38.55099
## 27 79.78280 111.49984 100.57107 65.70230 82.87129 76.05717  76.43512 84.61914
##           9       10
## 11 92.02057 60.94443
## 15 98.74849 75.17793
## 21 59.69994 45.00677
## 27 79.22119 87.29008
##
## $advance
##           1        2        3        4        5        6        7        8
## 13 35.20979 35.40488 16.41083 18.41917 19.36118 22.67411 13.89059 35.01444
##           9      10
## 13 38.84869 29.8599
```

Not very useful for viewing the data, but it lets us harness the `mice` package functions for analyzing the data and pooling the results.

The mice workflow

The `mice` package workflow is wonderfully streamlined and is how I usually do multiple imputation in practice. There are functions and objects for each step of the process:

1. Take your data.frame with missing data and impute the missing value with the `mice` function that returns a `mids` object (i.e., **m**ultiply **i**mputed **d**ata **s**et).
2. Take your `mids` object and analyze each imputed dataset with the `mids` method for the `with` function that returns a `mira` object (i.e., **m**ultiply **i**mputed **r**epeated **a**nalysis).
3. Take your `mira` object and pool the results from each repeated analysis with the `pool` function that returns a `mipo` object (i.e., **m**ulply **i**mputed **p**ooled **o**utcome).
4. Take your `mipo` object and summarize the results with the `mipo` method for the `summary` function that returns a `mipo.summary` object.

We will only be doing steps 2, 3, and 4 for analyzing and pooling. We start with step 2. Remember the `with` function expects non-standard evaluation.

```r
attitude_mira <- with(data = attitude_mids, expr = { # with.mids
   lm(rating ~ complaints + privileges + learning + raises + critical + advance)
})
summary.default(attitude_mira)
```

```
##          Length Class  Mode
## call      3     -none- call
## call1     7     -none- call
## nmis      7     -none- numeric
## analyses 10     -none- list
```

A `mira` object is a list with 4 elements. The most important one is the fourth "analyses" element which contains a list of model objects - one for each imputed dataset.

```r
head(attitude_mira[["analyses"]], n = 3)
```

```
## [[1]]
##
## Call:
## lm(formula = rating ~ complaints + privileges + learning + raises +
##     critical + advance)
##
## Coefficients:
## (Intercept)   complaints   privileges     learning       raises     critical
##    4.057836     0.661330    -0.030368     0.214943    -0.005467     0.175102
##     advance
##   -0.143164
##
##
## [[2]]
##
## Call:
## lm(formula = rating ~ complaints + privileges + learning + raises +
##     critical + advance)
##
## Coefficients:
## (Intercept)   complaints   privileges     learning       raises     critical
##     8.30508      0.66568     -0.14211      0.14269      0.06215      0.13510
##     advance
##    -0.05963
##
##
## [[3]]
##
## Call:
## lm(formula = rating ~ complaints + privileges + learning + raises +
##     critical + advance)
##
## Coefficients:
## (Intercept)   complaints   privileges     learning       raises     critical
##    4.222265     0.643043     0.094448     0.282931    -0.005408     0.110778
##     advance
##   -0.276319
```

This is the same as the `lm_imp` object we created in the manual calculations. Now we proceed to step 3 which will pool the results:

```
attitude_mipo <- pool(attitude_mira)
summary.default(attitude_mipo)
```

```
##         Length Class      Mode
## call     2     -none-     call
## m        1     -none-     numeric
## pooled  11     data.frame list
## glanced 12     data.frame list
```

A `mipo` object is another list with 4 elements. The most important one is the third "pooled" element which contains a data.frame of the pooled coefficients, sampling variances, and degrees of freedom as well as the fraction of missing information (FMI).

```
print(attitude_mipo[["pooled"]])
```

```
##          term  m    estimate         ubar            b            t dfcom
## 1 (Intercept) 10  8.25122733 129.78697119 20.662380773 152.51559004    23
## 2  complaints 10  0.63116347   0.02765362  0.002228301   0.03010475    23
## 3  privileges 10 -0.05715522   0.01687296  0.008205213   0.02589870    23
## 4    learning 10  0.17977462   0.02864853  0.003734721   0.03275673    23
## 5      raises 10  0.05618392   0.05440959  0.007060274   0.06217589    23
## 6    critical 10  0.14488926   0.01841712  0.008733128   0.02802356    23
## 7     advance 10 -0.16062333   0.02905430  0.005116761   0.03468274    23
##         df        riv     lambda       fmi
## 1 17.29578 0.17512250 0.14902489 0.2328822
## 2 19.22598 0.08863689 0.08142007 0.1640783
## 3 11.65611 0.53492289 0.34850148 0.4374062
## 4 17.98450 0.14339976 0.12541525 0.2087706
## 5 17.99915 0.14273775 0.12490858 0.2082540
## 6 11.80268 0.52160379 0.34279869 0.4315936
## 7 16.90554 0.19372130 0.16228352 0.2464527
```

While it contains all the statistical information we need to obtain p-values and confidence intervals, it does not directly provide them. We proceed to step 4 to get them:

```
tmp <- summary(attitude_mipo, type = "all", conf.int = T, conf.level = 0.95) # summary.mip
result_pool3 <- pick(tmp, c("m","ubar","b","t","dfcom"), nm = T, not = T)
print(result_pool3, digits = 3)
```

```
##          term estimate std.error statistic   df p.value   2.5 % 97.5 %    riv
## 1 (Intercept)   8.2512    12.350     0.668 17.3 0.51287 -17.771 34.273 0.1751
## 2  complaints   0.6312     0.174     3.638 19.2 0.00172   0.268  0.994 0.0886
## 3  privileges  -0.0572     0.161    -0.355 11.7 0.72882  -0.409  0.295 0.5349
## 4    learning   0.1798     0.181     0.993 18.0 0.33374  -0.200  0.560 0.1434
## 5      raises   0.0562     0.249     0.225 18.0 0.82427  -0.468  0.580 0.1427
## 6    critical   0.1449     0.167     0.866 11.8 0.40401  -0.221  0.510 0.5216
## 7     advance  -0.1606     0.186    -0.862 16.9 0.40049  -0.554  0.232 0.1937
##   lambda   fmi
## 1 0.1490 0.233
```

```
## 2 0.0814 0.164
## 3 0.3485 0.437
## 4 0.1254 0.209
## 5 0.1249 0.208
## 6 0.3428 0.432
## 7 0.1623 0.246
```

Now we have the p-values and confidence intervals. We also have some new columns: "riv" and "lambda". "riv" is the relative increase in variance we introduced earlier. "lambda" is the unadjusted FMI, while "fmi" is the adjusted FMI.

Comparison to manual calculations

Let's compare the statistical information from `mice` to try to reconcile the differences across the manual calculations and `mitools` results: 1) degrees of freedom and 2) fraction of missing information.

```
print(result_pool["df"]) # manual calculations
```

```
##                   df
## (Intercept) 17.29578
## complaints  19.22598
## privileges  11.65611
## learning    17.98450
## raises      17.99915
## critical    11.80268
## advance     16.90554
```

```
print(result_pool2["df"]) # mitools R package
```

```
##                   df
## (Intercept) 17.52304
## complaints  19.36730
## privileges  11.97585
## learning    18.18532
## raises      18.19936
## critical    12.12224
## advance     17.14598
```

```
print(result_pool3[c("term","df")]) # mice R package
```

```
##          term       df
## 1 (Intercept) 17.29578
## 2  complaints 19.22598
## 3  privileges 11.65611
## 4    learning 17.98450
## 5      raises 17.99915
## 6    critical 11.80268
## 7     advance 16.90554
```

We see that the degrees of freedom from the `mice` results are the same as our manual calculations. So it is the `mitools` df values that are in the minority. We already described how the df formulas differ earlier; however,

another way to think about the difference is that the `mitools` package does not include the uncertainty due to finite sampling in one part of the df formula. Here is our manual calculation (and `mice`) formula:

$$df_{imputation} = \frac{1}{\frac{1}{\frac{m-1}{FMI^2}} + \frac{1}{(1-FMI)\left(\frac{df_{complete}+1}{df_{complete}+3}\right)df_{complete}}}$$

Instead of $(1 - FMI)$, the `mitools` formula has $\left(1 - \frac{SE^2_{missing.data}}{SE^2_{sample.size} + SE^2_{missing.data}}\right)$.

```
df_com <- df.residual(lm_imp[[1]])
v <- (m - 1) / fmi^2
vhat2 <- (1 - (diag(missing_data) / (diag(sample_size) + diag(missing_data)))) *
   ((df_com + 1) / (df_com + 3)) * df_com
df_imp_mitools <- (v^(-1) + vhat2^(-1))^(-1)
print(df_imp_mitools)
```

```
## (Intercept)  complaints  privileges    learning      raises    critical
##    17.52304    19.36730    11.97585    18.18532    18.19936    12.12224
##     advance
##    17.14598
```

Again, we get the same degrees of freedom as the `mitools` package. Let's move on to the fraction of missing information:

```
result_pool["fmi"] # manual calculations
```

```
##                     fmi
## (Intercept) 0.14902489
## complaints  0.08142007
## privileges  0.34850148
## learning    0.12541525
## raises      0.12490858
## critical    0.34279869
## advance     0.16228352
```

```
v2d(fmi_adj, rtn.dim.nm = "fmi_adj") # manual calculations
```

```
##                fmi_adj
## (Intercept) 0.1531938
## complaints  0.0827703
## privileges  0.3654010
## learning    0.1284563
## raises      0.1279269
## critical    0.3593137
## advance     0.1671435
```

```
result_pool2["fmi"] # mitools R package
```

```
##                     fmi
## (Intercept) 0.2319536
## complaints  0.1635560
```

```
## privileges  0.4355080
## learning    0.2079804
## raises      0.2074669
## critical    0.4297172
## advance     0.2454482
```

```
result_pool3[c("term","lambda")] # mice R package
```

```
##          term     lambda
## 1 (Intercept) 0.14902489
## 2  complaints 0.08142007
## 3  privileges 0.34850148
## 4    learning 0.12541525
## 5      raises 0.12490858
## 6    critical 0.34279869
## 7     advance 0.16228352
```

```
result_pool3[c("term","fmi")] # mice R package
```

```
##          term       fmi
## 1 (Intercept) 0.2328822
## 2  complaints 0.1640783
## 3  privileges 0.4374062
## 4    learning 0.2087706
## 5      raises 0.2082540
## 6    critical 0.4315936
## 7     advance 0.2464527
```

We see the FMI from the `mice` results is different than both the manual calculations and `mitools` results! Ugh - more confusion… let's see if we can figure out what is causing the differences. It seems that the `mice` package uses a slightly different formula for the adjusted FMI. The formula includes the df, which is largely a function of the sample size:

$$FMI_{adjusted} = FMI + \frac{\frac{2*SE^2_{sample.size}}{df_{imputation}+3}}{SE^2_{total.error}}$$

```
fmi_adj_mice <- fmi + (((2 * diag(sample_size)) / (df_imp + 3)) / diag(total_error))
print(fmi_adj_mice)
```

```
## (Intercept)  complaints  privileges    learning      raises    critical
##   0.2328822   0.1640783   0.4374062   0.2087706   0.2082540   0.4315936
##     advance
##   0.2464527
```

Now we get the same adjusted FMI as the `mice` package. Let's try the same formula with the `mitools` df, since it could be the adjusted FMI formula is the same in the `mitools` and `mice` package and the values are only different due to the different df.

```
fmi_adj_mitools <- fmi + (((2 * diag(sample_size)) / (df_imp_mitools + 3)) / diag(total_error))
print(fmi_adj_mitools)
```

```
## (Intercept)  complaints  privileges    learning      raises    critical
##   0.2319536   0.1635560   0.4355080   0.2079804   0.2074669   0.4297172
##     advance
##   0.2454482
```

We see that the formula for the adjusted FMI is the same for both the `mitools` and `mice` package. So it is the manual calculation adjusted FMI values that are in the minority, which is based on Enders, 2010. I find it hard to believe that either Craig Enders or Stef van Buuren (the author of the `mice` R package) are "wrong" about anything related to missing data, so this might be a case where there is genuine disagreement about how to compute the adjusted FMI. The `mitools` and `mice` adjusted FMI asymptotically approaches the unadjusted FMI when the number of imputations *and* sample size approaches `Inf`. Personally, I find it strange that the FMI would have a sample size penalty, but maybe I'm just psychological attached to my manual calculations . Regardless, it seems that the best way to calculate the FMI is still being established and may be a topic of future statistical research (e.g., Pan, Wei, & Zou, 2018). With larger sample sizes the differences among the various types of adjusted FMIs should become more negligable.

As a summary, let's compare our final results computed the three different ways:

```
print(result_pool) # manual calculations
```

```
##                    est         se          t       df           p         lwr
## (Intercept)  8.25122733 12.3497202  0.6681307 17.29578 0.512868414 -17.7705033
## complaints   0.63116347  0.1735072  3.6376787 19.22598 0.001724592   0.2682974
## privileges  -0.05715522  0.1609307 -0.3551542 11.65611 0.728817486  -0.4089439
## learning     0.17977462  0.1809882  0.9932947 17.98450 0.333743577  -0.2004910
## raises       0.05618392  0.2493509  0.2253207 17.99915 0.824268278  -0.4676847
## critical     0.14488926  0.1674024  0.8655149 11.80268 0.404006318  -0.2205267
## advance     -0.16062333  0.1862330 -0.8624858 16.90554 0.400490198  -0.5537080
##                    upr        fmi   fmi_adj        riv    riv_adj
## (Intercept) 34.2729579 0.14902489 0.1531938 0.17512250 0.18090769
## complaints   0.9940295 0.08142007 0.0827703 0.08863689 0.09023945
## privileges   0.2946335 0.34850148 0.3654010 0.53492289 0.57579827
## learning     0.5600402 0.12541525 0.1284563 0.14339976 0.14738935
## raises       0.5800526 0.12490858 0.1279269 0.14273775 0.14669293
## critical     0.5103052 0.34279869 0.3593137 0.52160379 0.56082614
## advance      0.2324613 0.16228352 0.1671435 0.19372130 0.20068709
```

```
print(result_pool2) # mitools R package
```

```
##                    est         se          t       df           p         lwr
## (Intercept)  8.25122733 12.3497202  0.6681307 17.52304 0.512755330 -17.7705033
## complaints   0.63116347  0.1735072  3.6376787 19.36730 0.001708144   0.2682974
## privileges  -0.05715522  0.1609307 -0.3551542 11.97585 0.728650431  -0.4089439
## learning     0.17977462  0.1809882  0.9932947 18.18532 0.333600117  -0.2004910
## raises       0.05618392  0.2493509  0.2253207 18.19936 0.824240578  -0.4676847
## critical     0.14488926  0.1674024  0.8655149 12.12224 0.403560564  -0.2205267
## advance     -0.16062333  0.1862330 -0.8624858 17.14598 0.400323252  -0.5537080
##                    upr       fmi
## (Intercept) 34.2729579 0.2319536
## complaints   0.9940295 0.1635560
## privileges   0.2946335 0.4355080
## learning     0.5600402 0.2079804
## raises       0.5800526 0.2074669
```

```
## critical     0.5103052 0.4297172
## advance      0.2324613 0.2454482
```

```
print(result_pool3) # mice R package
```

```
##          term    estimate  std.error  statistic       df     p.value
## 1 (Intercept)  8.25122733 12.3497202  0.6681307 17.29578 0.512868414
## 2  complaints  0.63116347  0.1735072  3.6376787 19.22598 0.001724592
## 3  privileges -0.05715522  0.1609307 -0.3551542 11.65611 0.728817486
## 4    learning  0.17977462  0.1809882  0.9932947 17.98450 0.333743577
## 5      raises  0.05618392  0.2493509  0.2253207 17.99915 0.824268278
## 6    critical  0.14488926  0.1674024  0.8655149 11.80268 0.404006318
## 7     advance -0.16062333  0.1862330 -0.8624858 16.90554 0.400490198
##         2.5 %     97.5 %        riv     lambda       fmi
## 1 -17.7705033 34.2729579 0.17512250 0.14902489 0.2328822
## 2   0.2682974  0.9940295 0.08863689 0.08142007 0.1640783
## 3  -0.4089439  0.2946335 0.53492289 0.34850148 0.4374062
## 4  -0.2004910  0.5600402 0.14339976 0.12541525 0.2087706
## 5  -0.4676847  0.5800526 0.14273775 0.12490858 0.2082540
## 6  -0.2205267  0.5103052 0.52160379 0.34279869 0.4315936
## 7  -0.5537080  0.2324613 0.19372130 0.16228352 0.2464527
```

All the coefficients and standard errors are the same, while the degrees of freedom, p-values, and FMI slightly differ due to the different formulas. While dissatisfying, I find these differences rarely change my statistical inference. Afterall, the p-values are all equal to the third decimal place. However, there is something else that I have found to potentially change my statistical inference…

Impact of the number of imputed datasets

While traditional articles and chapters on multiple imputation will say you usually only need 5 or 10 imputed datasets, I often find it useful to use 100+ imputed datasets. That is because I am often interested in getting consistent p-values, which are stable across different multiple imputation runs. In social science, re-running analyses for revise and resubmits are the norm, rather than the exception. And if you only use 5-10 imputed datasets, you can re-run the analyses for your revise and resubmit and get different statistical inference from p-values dancing around the alpha cutoff (e.g., 0.05). Instead, I use 100, 250, or even 500 imputed datasets so that my p-values are consistent to the third decimal.

To show you what I mean, I will re-create the imputed datasets 3 times (with 10 imputed datasets) and then re-run the `mice` analyses for each. To be clear, the difference in the final results is due to the change in the imputed values since creating multiply imputed data involves a random process that will depend on the random seed we give R.

p-values

First, I will create a function for extracting the p-values. What this function will do is take a 1) given number of imputed datasets `m` and 2) seed value `seed` to capture the random process, and then return the vector of p-values from the linear regression.

```
get_pvalues <- function(m, seed) {
   mids <- mice(attitude_na, m = m, method = "norm", seed = seed, printFlag = FALSE)
   mira <- with(mids, # with.mids
      expr = lm(rating ~ complaints + privileges + learning + raises + critical + advance))
```

```
   mipo <- pool(mira)
   smry <- summary(mipo) # summary.mipo
   p <- setNames(smry[["p.value"]], nm = smry[["term"]])
   return(p)
}
```

Now we call the function 3 times and compare the p-values to the original ones:

```
tmp <- lapply(X = c(020152022, 03222023, 04302024),
   FUN = function(seed) get_pvalues(m = 10, seed = seed))
pvalues_10 <- lv2d(tmp, along = 2)
cbind(pvalues_10, after = 0, col.nm = "original") <- result_pool3[["p.value"]]
print(pvalues_10)
```

```
##                original         1           2           3
## (Intercept) 0.512868414 0.3538382 0.361394980 0.4380563449
## complaints  0.001724592 0.0129851 0.001620315 0.0009318646
## privileges  0.728817486 0.7578593 0.610839414 0.6503863151
## learning    0.333743577 0.4665363 0.347352028 0.4017467992
## raises      0.824268278 0.6641562 0.723426642 0.7771289143
## critical    0.404006318 0.6641102 0.522719046 0.5923997699
## advance     0.400490198 0.5254053 0.478804300 0.5625805018
```

As you can see, the p-values dance around a decent amount. If we were using an alpha level of .01, our statistical inference would actually be different for the first re-run analysis. In that case, we would have to change our discussion section to reflect the re-run analysis. Rather dissatisfying if you ask me. Even if we use the more conventional .05 as our alpha level, we would still need to change the asterixis in our regression table. Let's see what happens when we up the number of imputed datasets to 100.

```
tmp <- lapply(X = c(020152022, 03222023, 04302024),
   FUN = function(seed) get_pvalues(m = 100, seed = seed))
pvalues_100 <- lv2d(tmp, along = 2)
print(pvalues_100)
```

```
##                      1           2           3
## (Intercept) 0.424763983 0.466098185 0.468220193
## complaints  0.003042423 0.003464489 0.001316272
## privileges  0.756844450 0.870009293 0.687495003
## learning    0.347605825 0.403608085 0.368012962
## raises      0.765296814 0.716029011 0.784547303
## critical    0.536473797 0.520865446 0.471571318
## advance     0.468270888 0.446831472 0.458783535
```

More similar p-values, but still some dancing around a bit. Let's go up to 250 imputed datasets.

```
tmp <- lapply(X = c(020152022, 03222023, 04302024),
   FUN = function(seed) get_pvalues(m = 250, seed = seed))
pvalues_250 <- lv2d(tmp, along = 2)
print(pvalues_250)
```

```
##                         1           2           3
## (Intercept) 0.446137487 0.474477783 0.469419309
## complaints  0.001820063 0.001859328 0.001817972
## privileges  0.798183028 0.732789985 0.753232440
## learning    0.382059463 0.358400744 0.323411436
## raises      0.796152457 0.798041622 0.788430349
## critical    0.512670469 0.451508305 0.489259626
## advance     0.459331601 0.472671496 0.443525518
```

Much more similar p-values now. You will notice the larger p-values still dance around a bit, but the smaller p-values - that have bearing on statistical inference - are stable to three decimal places. When you think about the sampling error due to missing data as a variance where the number of imputed datasets is the "sample size", you can see why there is quite a bit of noise from only 5 to 10 imputed datasets. Any variance estimated with 5 to 10 observations is going to be unstable. But a variance with 250 observations - much more stable.

Statisticians have noticed this issue and come up with rules of thumb around how many imputed datasets are needed for consistent results (e.g., Table 3; Bodner, 2008). However, I have personally have found those rules of thumb to not be large enough (e.g., 50 imputed datasets for FMI = 0.50). A rule of thumb I have started using is multplying your percentage of missing data by 5 and then having that be your number of imputed datasets. Obviously, the univariate percentage of missing data differs by variable, so I will determine what the multivariate percentage of missing data is for the listwise deletion version of my analysis and go off that. So if have only 1% missing data, then I only need 5 imputed datasets; if I have 20% missing data, then I will use 100 imputed datasets; if I have 50% missing data, then I will use 250 imputed datasets. In our example dataset, our multivariate missingness from listwise deletion is , so I would have used - rounded to the nearest integer - imputed datasets.

Keep in mind that we would never expect p-values to replicate to the third decimal and p-values dance around a lot more due to random chance than we humans expect (pg. 130; Cumming, 2012). If we were to take a new random sample of n = 30 from a population, the p-values would likely not even replicate to the first decimal. Even with n = 300 or n = 3000, our p-values still would unlikely replicate to the third decimal. However, for the practical issue of writing-up findings from a single sample for an academic journal, it is nice to have consistent numbers and interpretations across re-runs of the analysis. In sum, increasing the number of imputed datasets can help maintain consistent p-values across re-runs of your analysis.

confidence intervals

Because I have mixed feelings about focusing so much on p-values, I thought I would also show the consistency of confidence intervals when re-running analyses that use multiple imputation. Personally, I am not a big fan of statistical inference based on alpha level cutoffs for p-values (i.e., NHST; **N**ull **H**ypothesis **S**ignificant **T**esting). The dichotomization of statistical evidence looses information and results in dissatisfying interpretations for p-values just above or below the alpha level. Instead, I prefer to think about p-values as a continuous index of statistical evidence. From this point of view, a p-value of .06 in one re-run of the analysis and a p-value of .04 from another re-run of the analysis provide essentially the same statistical information. However, most co-authors, reviewers, and editors I work with expect NHST where the alpha level is emphasized (e.g., .05).

Even if I am working with colleagues we share my prefered perspective on p-values, the p-value is not that informative of statistical evidence. On it's own, it doesn't tell us any information about the effect size and is highly dependent on the sample size. For these reasons, I prefer confidence intervals. Obviously, if you focus only on whether the confidence interval includes zero at a certain alpha level, then the same dichotomization of statistical evidence happens. However, if you use confidence intervals to quantity uncertainty of estimation (Cumming, 2012), then you have a more informative continous index of statistical evidence [2].

[2]It is worth mentioning that confidence intervals around uninterpretable effect sizes are not very informative. For example,

Let's try 3 different random seeds with 10 imputed datasets. First the function to return the confidence intervals.

```
get_ci <- function(m, seed) {
   mids <- mice(attitude_na, m = m, method = "norm", seed = seed, printFlag = FALSE)
   mira <- with(mids, # with.mids
      expr = lm(rating ~ complaints + privileges + learning + raises + critical + advance))
   mipo <- pool(mira)
   smry <- summary(mipo, conf.int = TRUE) # summary.mipo
   ci <- smry[c("2.5 %","97.5 %")]
   row.names(ci) <- smry[["term"]]
   return(ci)
}
```

Now we call the function 3 times and see that the confidence intervals also dance around quite a bit:

```
tmp <- lapply(X = c(020152022, 03222023, 04302024),
   FUN = function(seed) get_ci(m = 10, seed = seed))
ci_10 <- ld2a(tmp)
print(ci_10)
```

```
## , , 1
##
##                   2.5 %     97.5 %
## (Intercept) -14.7787257 39.2149428
## complaints    0.1435451  1.0127484
## privileges   -0.3881711  0.2889009
## learning     -0.3027137  0.6261358
## raises       -0.4449841  0.6802398
## critical     -0.3117808  0.4720061
## advance      -0.5729671  0.3042461
##
## , , 2
##
##                   2.5 %     97.5 %
## (Intercept) -13.7544739 36.0215827
## complaints    0.2613537  0.9590399
## privileges   -0.4330007  0.2654035
## learning     -0.2306208  0.6158798
## raises       -0.4267435  0.6029228
## critical     -0.2085924  0.3967402
## advance      -0.5445935  0.2678847
##
## , , 3
##
##                   2.5 %     97.5 %
## (Intercept) -16.6847845 36.8781522
## complaints    0.2961171  0.9863955
## privileges   -0.3883874  0.2500701
## learning     -0.2514527  0.5928425
```

when your effect size is an unstandardized regression coefficient that you don't know how to interpret. If you can't interpret the lower and upper boundary values of the confidence interval, then they don't add much information. In these instances, I often wonder if a p-value is equally (un)informative.

```
## raises       -0.4532474  0.5969020
## critical     -0.2421780  0.4104398
## advance      -0.5070054  0.2854654
```

Let's compare those to our original `mice` confidence intervals

```
print(result_pool3[c("term","2.5 %","97.5 %")])
```

```
##          term       2.5 %     97.5 %
## 1 (Intercept) -17.7705033 34.2729579
## 2  complaints   0.2682974  0.9940295
## 3  privileges  -0.4089439  0.2946335
## 4    learning  -0.2004910  0.5600402
## 5      raises  -0.4676847  0.5800526
## 6    critical  -0.2205267  0.5103052
## 7     advance  -0.5537080  0.2324613
```

Really only the first significant digit is consistent across re-runs of the analysis. Let's try 100 imputed datasets.

```
tmp <- lapply(X = c(020152022, 03222023, 04302024),
   FUN = function(seed) get_ci(m = 100, seed = seed))
ci_100 <- ld2a(tmp)
print(ci_100)
```

```
## , , 1
##
##                    2.5 %     97.5 %
## (Intercept) -15.8169465 35.9946843
## complaints    0.2378999  0.9883071
## privileges   -0.4134264  0.3079977
## learning     -0.2189063  0.5884125
## raises       -0.4472229  0.5981616
## critical     -0.2343390  0.4337996
## advance      -0.5331937  0.2555007
##
## , , 2
##
##                    2.5 %     97.5 %
## (Intercept) -16.9430798 35.5793073
## complaints    0.2276508  0.9834787
## privileges   -0.4129763  0.3543047
## learning     -0.2419235  0.5729718
## raises       -0.4367059  0.6235293
## critical     -0.2412881  0.4567916
## advance      -0.5576841  0.2574638
##
## , , 3
##
##                    2.5 %     97.5 %
## (Intercept) -16.4101277 34.3531493
## complaints    0.2821402  0.9879319
```

```
## privileges   -0.3799659  0.2573457
## learning     -0.2296267  0.5857655
## raises       -0.4418458  0.5767017
## critical     -0.2146593  0.4438792
## advance      -0.5380265  0.2540604
```

Greater simiarity across random seeds but still only the first significant digit is the consistent. And now with 250 imputed datasets.

```
tmp <- lapply(X = c(020152022, 03222023, 04302024),
   FUN = function(seed) get_ci(m = 250, seed = seed))
ci_250 <- ld2a(tmp)
print(ci_250)
```

```
## , , 1
##
##                   2.5 %     97.5 %
## (Intercept) -16.0438149 35.0356431
## complaints    0.2651394  0.9895576
## privileges   -0.3930072  0.3081233
## learning     -0.2347937  0.5818180
## raises       -0.4504604  0.5793088
## critical     -0.2338611  0.4496229
## advance      -0.5397870  0.2547675
##
## , , 2
##
##                   2.5 %     97.5 %
## (Intercept) -16.7705437 34.7078967
## complaints    0.2629370  0.9831922
## privileges   -0.4069727  0.2937329
## learning     -0.2259475  0.5899724
## raises       -0.4501159  0.5774529
## critical     -0.2093170  0.4498052
## advance      -0.5349794  0.2589425
##
## , , 3
##
##                   2.5 %     97.5 %
## (Intercept) -16.5942829 34.6536338
## complaints    0.2641305  0.9814641
## privileges   -0.3986505  0.2953212
## learning     -0.2062244  0.5893809
## raises       -0.4441413  0.5768014
## critical     -0.2188907  0.4387057
## advance      -0.5367466  0.2458238
```

Now most of the confidence intervals are consistent for two significant digits. Yet, not all of them (e.g., lower bound for "learning"); so, we would still need to update the confidence intervals in our regression table when we re-run our analyses for a revise and resubmit to a journal. Again, we would never expect confidence intervals to actually replicate to the second significant digit when randomly sampling from a population; however, having consistent confidence intervals across re-runs of analyses with multiply imputed data makes academic publishing a little simplier for us [3].

[3] I did not show this, but another quantity that is rather inconsistent across random seeds for multiply imputed dataset

Limitations

Analyzing and pooling results from multiply imputed data is a great way to get more accurate estimation and inferences in the presence of missing data. However, it is not without its limitations. The mathematical formulas used for pooling require a standard error and assume that sampling distributions are normally distributed. If either of those are not the case, then things get more complicated. For example, it isn't clear how to pool the results from chi-square tests of independence as there is no standard error associated with the statistical test. Same with indirect effects from mediation where we often use bootstrapping to compute non-normal confidence intervals. There are sometimes work arounds (e.g., using Monte Carlo confidence intervals instead of bootstrapped confidence intervals for indirect effects; Preacher & Selig, 2012), but other times I am simply not aware of any way to pool results and have to fall back on listwise deletion. In the years to come, I am sure statisticians will develop and disseminate pooling options for analyses that do not provide standard errors or have non-normal sampling distributions. Be on the look out for a future blog post on that topic.

References

Bodner, T. E. (2008). What improves with increased missing data imputations? *Structural Equation Modeling: A Multidisciplinary Journal, 15*(4), 651-675.

Cumming, G. (2012). *Understanding The New Statistics*. New York, NY: Taylor & Francis Group.

Enders, C. E. (2010). *Applied Missing Data Analysis*. New York, NY: Guilford Press.

Pan, Q., & Wei, R. (2018). Improved methods for estimating fraction of missing information in multiple imputation. *Cogent Mathematics & Statistics, 5*(1), 1551504.

Preacher, K. J., & Selig, J. P. (2012). Advantages of Monte Carlo confidence intervals for indirect effects. *Communication Methods and Measures, 6*(2), 77-98.

creation is the fraction of missing information (FMI). Obtaining consistent FMI values to the second decimal can actually require many more imputed datasets than is required for consistent p-values to the second decimal (e.g., 847; Bodner, 2008)