Missing Data & Cleaning

Joshua F. Wiley

2019-05-19

Contents

Download the raw `R` markdown code here `https://jwiley.github.io/MonashHonoursStatistics/Data_Missing.rmd`.

```
options(digits = 2)

## There are two new packages: (1) mice (2) VIM
## both are used for missing data. You can try
## installing by uncommenting the
## install.packages() code below

# install.packages('mice', type = 'binary')
# install.packages('VIM', type = 'binary')

## some people report also needing the zip pkg
## install.packages('zip', type = 'binary')

## some people also report type='binary' does
## not work if that happens to you, try:
## install.packages('mice')
## install.packages('VIM')

## once installed, open the packages using the
## library() calls below

library(data.table)
```

```
library(mice)
library(VIM)
library(ggplot2)

## read in the dataset
d <- readRDS("aces_daily_sim_processed.RDS")
```

1 Data Cleaning

A common data cleaning task is to identify and address extreme values. In multilevel / repeated measures data, extreme values may exist at the between person level (i.e., a person who is relatively extreme compared to other people) or at the within person level (i.e., an assessment that is relatively extreme for that person).

1.1 Between Person Cleaning

First we will look at the between person level. To do this, we need to create between person variables and then make sure we only have one row per person.

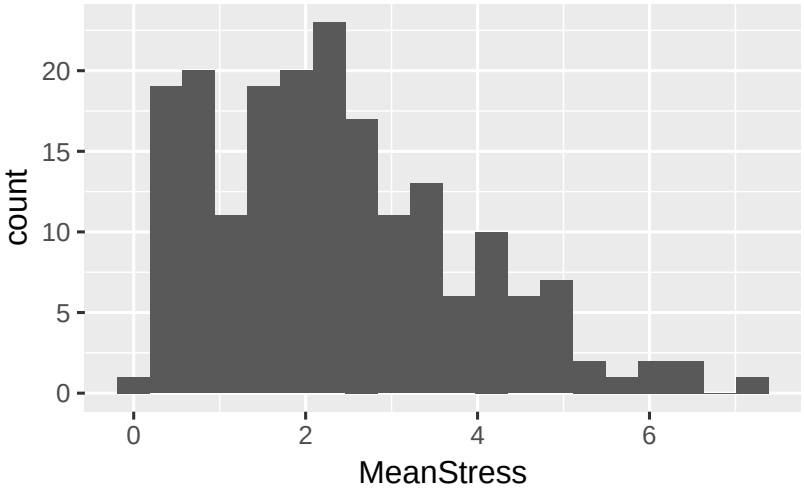
```
## make averages
d[, `:=`(MeanStress, mean(STRESS, na.rm = TRUE)),
    by = UserID]
## between person dataset (one row per person)
d.b <- d[!duplicated(UserID)]
```

Now we can process these between person data as any other dataset. One way to look for outliers is using z scores and histograms. We can create z scores in R using the `scale()` function which can scale variables to have mean 0 and standard deviation 1 by default or scale to any other values as desired. We can see the maximum z score and the graph also suggests that one person is a bit extreme with an average stress score above 6.

```
d.b[, `:=`(ZStress, scale(MeanStress))]

## summary of the data including min and max
summary(d.b$ZStress)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.
##    -1.5    -0.7    -0.1     0.0     0.7
##    Max.
##     3.4
```

```
ggplot(d.b, aes(MeanStress)) + geom_histogram(bins = 20)
```



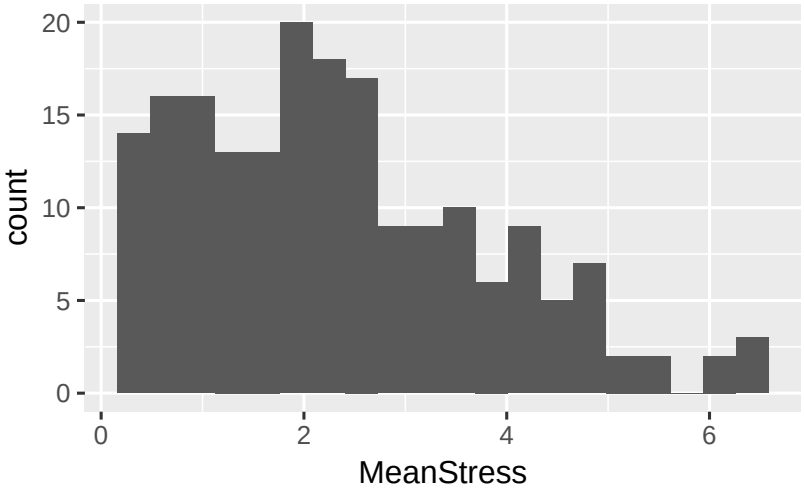
One option for extreme values is to winsorize them. Winsorizing basically involves setting a cutoff and then setting any scores above that cutoff at the cutoff or just slightly above it. Sometimes, people use z-scores to determine cut offs. However, these are themselves very sensitive to outliers. Percentiles tend to work better as they are not influenced by the presence of outliers. We can find percentiles in R using the `quantile()` function. For balance, it is important to winsorize both the top and bottom equally. Below we look at the extremes (0th and 100th percentiles), 1st and 99th percentiles, and median (50th percentile).

```
quantile(d.b$MeanStress, probs = c(0, 0.01, 0.5,
    0.99, 1), na.rm = TRUE)
```

```
##   0%   1%  50%  99% 100%
## 0.18 0.21 2.15 6.31 7.37
```

Suppose we wanted to winsorize the top and bottom 1 percent. We do it by selecting the extreme rows and altering just those values. Remaking the histogram we can see that the most extreme values have been pulled in slightly. We can also convert to z-scores again using `scale()` now that extreme values have been pulled in and summarize. Now the maximum is under 3 as well. Not required, but showing the extremes have been reduced.

```
d.b[MeanStress < 0.21, ':='(MeanStress, 0.21)]
d.b[MeanStress > 6.31, ':='(MeanStress, 6.31)]

ggplot(d.b, aes(MeanStress)) + geom_histogram(bins = 20)
```




```
summary(scale(d.b$MeanStress))
```

```
##        V1
##  Min.   :-1.47
##  1st Qu.:-0.74
##  Median :-0.14
##  Mean   : 0.00
##  3rd Qu.: 0.67
##  Max.   : 2.72
```

1.2 Wihin Person Cleaning

Cleaning at the within person level is a bit more complex. First, we might create within person z scores. These are made by subtracting individuals' own means and dividing by individuals' own standard deviations, as different people might be more or less variable. Then we can make a summary and histogram. There are some extreme and rare values at the tails that we might want to pull in.

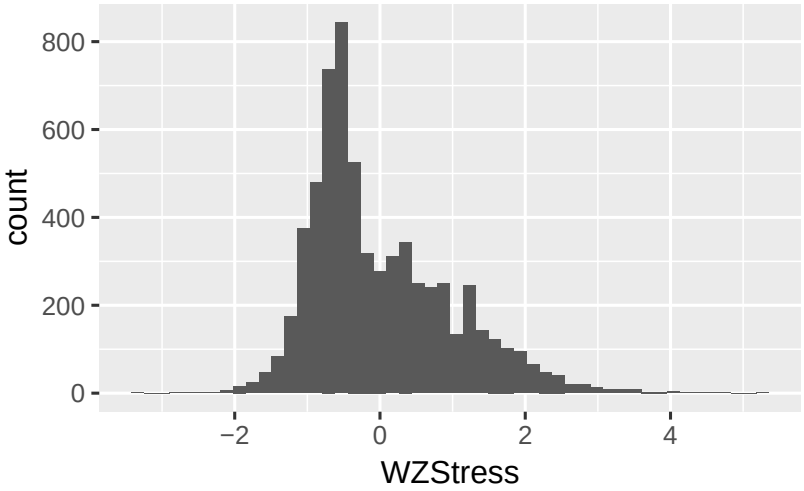
```
d[, ':='(WZStress, scale(STRESS)), by = UserID]
```

```
summary(d$WZStress)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.
##      -3      -1       0       0       1
##    Max.    NA's
##       5     525
```

```
ggplot(d, aes(WZStress)) + geom_histogram(bins = 50)
```

```
## Warning: Removed 525 rows containing non-finite
## values (stat_bin).
```



Winsorizing at the within level requires different values be applied to each person. To do this, we create new variables for each person.

```
d[, ':='(LLStress, quantile(STRESS, probs = 0.05,
    na.rm = TRUE)), by = UserID]
d[, ':='(ULStress, quantile(STRESS, probs = 0.95,
    na.rm = TRUE)), by = UserID]
```

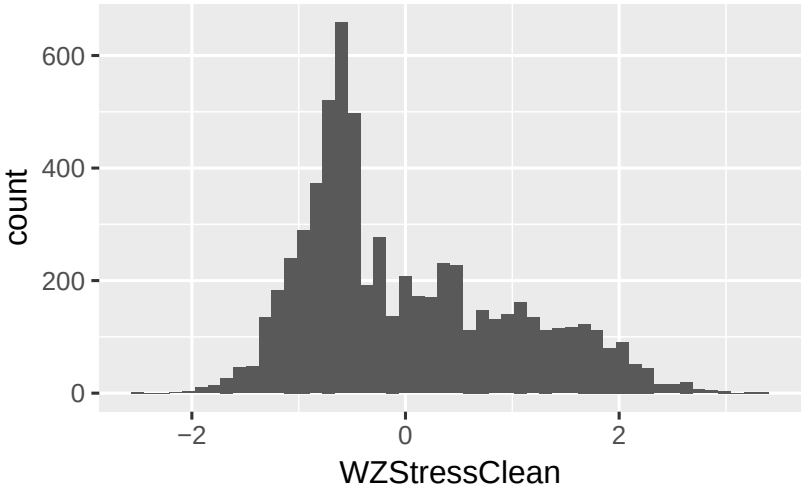
Now we can select any rows that exceed these values for each person, and set to their own person value. In this case, we will be winsorizing the top and bottom 5 percent for each person. The reason for choosing a higher value is that since each person only has on average about 30 observations, the 1st and 99th percentiles will be the extremes anyway. Afterward, we must again re-make the z-zscores and check the distribution.

```
d[, ':='(StressClean, STRESS)]
d[STRESS < LLStress, ':='(StressClean, LLStress)]
d[STRESS > ULStress, ':='(StressClean, ULStress)]

d[, ':='(WZStressClean, scale(StressClean)), by = UserID]

summary(d$WZStressClean)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.
##      -3      -1       0       0       1
##    Max.    NA's
##       3     525
```

```
ggplot(d, aes(WZStressClean)) + geom_histogram(bins = 50)
```

```
## Warning: Removed 525 rows containing non-finite
## values (stat_bin).
```



Even though we checked the between person level first. In general, it is better to clean the within person level first, because outliers on an individual assessment will bias the average for a person (i.e., the between person level).

1.3 Cleaning You Try It (Workbook)

Try to clean the within and between level of Negative Affect. Try these steps, as we did above.

- Calculate and plot individual within person z scores for Negative Affect (`NegAff`).
- Winsorize at the within person level and then re-calculate and plot within person z scores for Negative Affect
- Calculate average Negative Affect (make sure to use the "clean" version). Create a between person dataset with only one row per person. Examine whether there are between person outliers on negative affect, and if so, winsorize. Then check whether winsorizing at the percentile you chose was enough to remove apparent extreme values.

```
## within person

d[,  := scale(   ), by = UserID]

summary(d$   )

ggplot(d, aes(   )) +
  geom_histogram(bins = 50)
```

```
d[,   := quantile(   , probs = .05, na.rm = TRUE),
  by = UserID]
d[,    := quantile(   , probs = .95, na.rm = TRUE),
  by = UserID]

d[, Clean :=   ]
d[   <   , Clean :=   ]
d[   >   , Clean :=   ]

d[, WZClean := scale(   ), by = UserID]

summary(d$    )

ggplot(d, aes(    )) +
  geom_histogram(bins = 50)


## between person

## make averages
d[,     := mean(   , na.rm=TRUE), by = UserID]

## between person dataset (one row per person)
d.b <- d[!duplicated(UserID)]

d.b[,      := scale(    )]

## summary of the data including min and max
summary(d.b$   )

ggplot(d.b, aes(   )) +
  geom_histogram(bins = 20)


quantile(d.b$,
         probs = c(0, .01, .5, .99, 1),
         na.rm = TRUE)

d.b[    <   ,     :=   ]
d.b[    >   ,     :=   ]

ggplot(d.b, aes(    )) +
  geom_histogram(bins = 20)
```

```
summary(scale(d.b$    ))
```

```
## Error: <text>:4:6: unexpected assignment
## 3:
## 4: d[,  :=
##         ^
```

2 *Missing Data: Multiple Imputation*

Missing data are very common, but also problematic

- Results from the non-missing data may be biased
- Missing data cause a loss of efficiency

The easiest approach to "address" missing data is to only analyse complete cases (i.e., list-wise deletion). This often leads to inefficiency (e.g., discarding data on X and Z because observation on variable Y are missing). Also, list wise deletion may bias results unless the data are missing completely at random (MCAR).

Missing data often are classified:

- Missing completely at random (MCAR) when the missingness mechanism is completely independent of the estimate of our parameter(s) of interest
- Missing at random (MAR) when the missingness mechanism is *conditionally* independent of the estimate of our parameter(s) of interest
- Missing not at random (MNAR) when the missingness mechanism is associated with the estimate of our parameter(s) of interest

Our parameters may be anything, such as a mean, a standard deviation, a regression coefficient, etc. We will explore **multiple imputation (MI)** as one robust way to address missing data.

2.1 *Multiple Imputation (MI) Theory*

Multiple imputation is one robust way to address missing data. It involves generate multiple, different datasets where in each one, different, plausible values are imputed for the missing data, resulting in each imputed dataset have "complete" data since missing data are filled in by predictions.

As the problem with missing data is very generic and impacts every statistic, even the most basic decriptive statistics, we need a general language for talking about it.

- Let Q be some population value (e.g., a mean, a regression coefficient).

- Let $\hat{Q}$ be an estimate of Q with some estimate of uncertainty due to sampling variation, calculated typically in each imputed dataset.
- Let $\bar{Q}$ be the average of a set of estimates, $\hat{Q}$ across different imputed datasets, with some estimate of uncertainty both due to sampling variation impacting $\hat{Q}$ and missing data uncertainty (causing variation in $\hat{Q}$ from one imputed dataset to the next.

We will not discuss how to estimate $\hat{Q}$ as this is specific to the analysis being performed (e.g., mean, regression coefficient, mean difference from t-test, etc.). It does not really matter for the purpose of MI. All we need is an estimate and (valid) standard error.

The general steps in MI are as follows:

1. Start with the incomplete data (the raw dataset with missing data).
2. Generate m datasets with no missingness, by filling in *different* plausible values for any missing data. We will discuss this more later.
3. Perform the analysis of interest on **each** imputed dataset. That is, the analysis of interest is repeated m times. This generates m different $\hat{Q}$ estimates and associated standard errors.
4. Pool the results from the analyses run on each imputed dataset to generate an overall estimate, $\bar{Q}$.

Previously in statistics, we have grown accustomed to reporting $\hat{Q}$, whether that be a mean, correlation, regression coefficient, or any other statistic of interest. With missing data addressed using MI, we instead report $\bar{Q}$, which is defined as:

$$\bar{Q} = \frac{1}{m}\sum_{i=1}^{m}\hat{Q}_i$$

This is simply the average of the $\hat{Q}$ estimates from each individual imputed dataset. What is more unique and forms an important difference between multiple imputation and other simpler methods (e.g., single mean imputation) is the variance estimate.

Let $\hat{V}_i$ be the variance estimate (e.g., squared standard error) for the *ith* imputed dataset for $\hat{Q}_i$. Then:

$$\bar{V} = \frac{1}{m}\sum_{i=1}^{m}\hat{V}_i$$

$\bar{V}$ is the average uncertainty estimate of $\hat{Q}$ across the multiply imputed datasets.

However, there is another source of uncertainty in our estimate of $\bar{Q}$. The between imputed dataset variation, which is:

$$B = \frac{1}{m-1}\sum_{i=1}^{m}(\hat{Q}_i - \bar{Q})^2$$

B captures the variance in the estimates, $\hat{Q}$. Because the observed, non-missing data never change from one imputed dataset to another, the only reason that $\hat{Q}$ will change is when the plausible values imputed for the missing data change. Thus B is an estimate of the uncertainty in $\bar{Q}$ due to missing data.

Finally, for practical reasons we do not generate an infinite number of imputed datasets. We could simply continue multiply imputing more datasets, but this takes additional time. Because of this we do not have the *population* of imputed datasets, instead we have a (random) sample of all possible multiply imputed datasets. This results in some simulation error which is added to the uncertainty in our $\bar{Q}$ estimate. Specifically, this uncertainty is:

$$\frac{B}{m}$$

Now we finally have all the pieces to determine what the total uncertainty in our average estimate, $\bar{Q}$ is:

$$T = \bar{V} + B + \frac{B}{m}$$

Practically, this can be thought of as: the uncertainty in the estimate of our statistic of interest from multiply imputed data, $\bar{Q}$ is due to: (1) sampling variation $\bar{V}$, (2) uncertainty in how we imputed the missing data B, and (3) uncertainty because we did not generate an infinite number of imputed datasets, $\frac{B}{m}$. One implication is that we can reduce the uncertainty somewhat, simply by generating more imputed datasets. However, this yields diminishing returns and slows analysis (remember, the analysis of interest must be run separately **on each imputed dataset**. 100 imputed datasets would mean running an analysis 100 times, which takes more than running it for, say, 20 imputed datasets. In most cases, with modern computers, people recommend 25 - 100 imputed datasets now.

2.2 *Multiple Imputation Steps*

The general process to generate multiply imputed using a robust approach: fully conditional specification (FCS) or multiple imputation through chained equations (mice) is:

1. Start with a raw dataset with missing data
2. Fill in the missing data values with **initial** estimates[1].
3. For each variable that has missing data, build a prediction model from other variables[2].
4. Use the model to predict the missing data[3].
5. Repeat steps 2 and 3 until the predicted complete dataset does not vary within some tolerance from one iteration to the next,

[1] The first "fill in" estimates are generally fast and easy to generate. For example, fill in missing using the median for each variable or even random numbers within the observed data range.

[2] Typically, all other variables in the dataset being imputed are used in the prediction model, but it is possible to use a subset. The most common case is to use Generalized Linear Models (GLMs) as the prediction model, so linear regression for continuous outcomes, logistic regression for binary outcomes, etc. However, any prediction model can be used, including more complex methods such as machine learning.

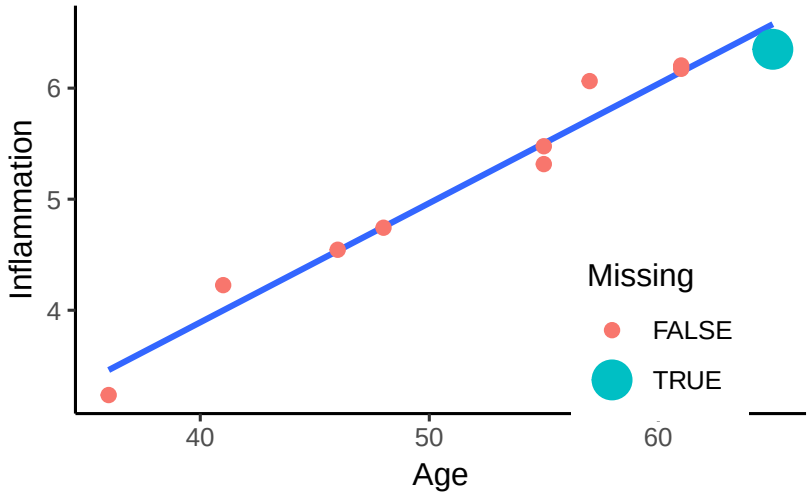
[3] At this step, the average / best pre-

indicating convergence.

These steps may be easier to understand with a concrete example. Here we setup a small dataset with age and inflammation.

```
set.seed(12345)
x <- data.frame(Age = sort(round(runif(10, 30,
    65))))
x$Inflammation <- rnorm(10, x$Age/10, sd = 0.2)
x$Missing <- as.logical(rep(FALSE:TRUE, c(9, 1)))

scatterp <- ggplot(x, aes(Age, Inflammation)) +
    stat_smooth(method = "lm", se = FALSE) + geom_point(aes(colour = Missing,
    size = Missing)) + theme_classic() + theme(legend.position = c(0.8,
    0.2))

print(scatterp)
```

```
## Warning: Using size for a discrete variable
## is not advised.
```



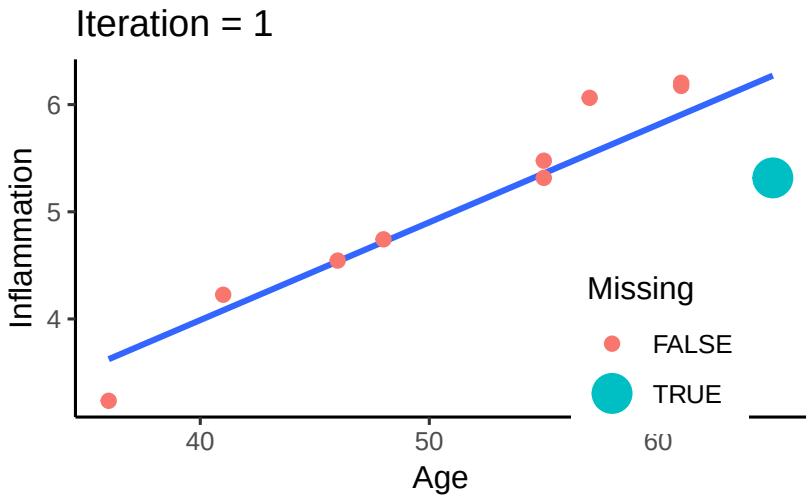

```
## create a missing dataset
y <- x
y$Inflammation[y$Missing] <- NA
```

Here is a for loop that iterates through filling in missing values, first using the median and then based on linear regression. The graphs following show the scatter plots with the filled in missing value at the first iteration and then later iterations[4]. Watch how the imputed missing value changes from one iteration to the next and how with later iterations, the amount of change gets smaller and smaller.

[4] The goal behind iterating is to build a prediction model to impute the missing data. However, building that model in the first place is difficult when the data on which the model would be built are (partially) missing. To deal with this, we iterate between predicting the missing data and (re)building a model on the new dataset, now that missingness has been imputed. Multiple iterations are required because every time we generate new predictions to fill in the missing data, the dataset on which the prediction model was built changes, so we must rebuild the prediction model using the latest

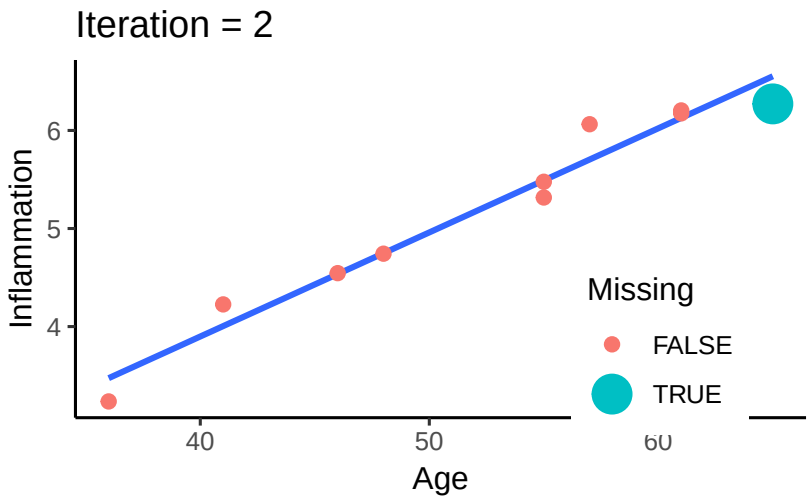
```
## create an imputed dataset
yimp <- y
p <- vector("list", length = 10)
imps <- vector("numeric", length = 10)

for (i in 1:10) {
    if (i == 1) {
        imps[i] <- median(y$Inflammation, na.rm = TRUE)
    } else if (i > 1) {
        m <- lm(Inflammation ~ Age, data = yimp)
        imps[i] <- fitted(m)[yimp$Missing]
    }
    yimp$Inflammation[yimp$Missing == TRUE] <- imps[i]
    p[[i]] <- scatterp %+% yimp + ggtitle(sprintf("Iteration = %d",
        i))
}

print(p[[1]])
```

```
## Warning: Using size for a discrete variable
## is not advised.
```



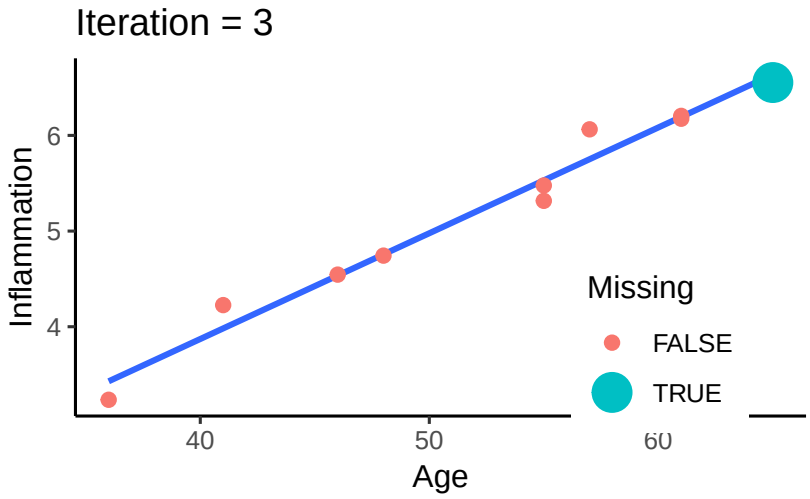

```
print(p[[2]])
```

```
## Warning: Using size for a discrete variable
## is not advised.
```



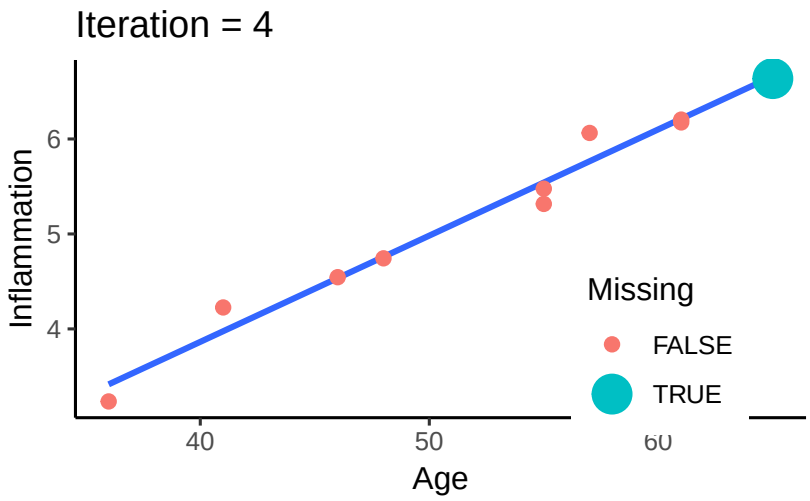

```
print(p[[3]])
```

```
## Warning: Using size for a discrete variable
## is not advised.
```




```
print(p[[4]])
```

```
## Warning: Using size for a discrete variable
## is not advised.
```



We can print out the imputed values too and see how those change. This highlights how over iterations, there are big changes at first, and with more iterations the changes are smaller and smaller. This is what is meant by convergence where you converge to a final, stable value. We could keep going by at some point we are close enough, we can stop. We might decide,say, that once it does not change beyond the 2nd decimal place, we stop.

```
print(imps, digits = 5)
```

```
##  [1] 5.3161 6.2708 6.5526 6.6358 6.6603
##  [6] 6.6676 6.6697 6.6703 6.6705 6.6706
```

This gets us through Steps 2 to 5. The final step is to use those final models to impute missing data[5], generating the desired number of imputed datasets (e.g., 100).

Often, the prediction models are generalized linear models (GLMs), such as linear regression. GLMs are familiar and fast, but there are drawbacks.

- GLMs assume linear relationships on the scale of the link function. This can be problematic for MI as if you are imputing, say, 50 variables it is very time consuming to manually check whether the assumption is met.
- GLMs only include interactions between variables when specified by the analyst. Often, analysts do not know which variables should interact. If there are, say, 50 variables, the possible number of interactions is overwhelming.
- In small datasets (e.g., 100 people) it is easily possible there may be more variables then people. GLMs require that the sample size be larger then the number of predictors, making them a poor choice for MI in these cases.

[5] This can be a rather complex step. Generating the average prediction from, say, a linear regression is straight forward. However, that average prediction does not take into account that the linear regression model is uncertain in its prediction. To take uncertainty into account, from parametric models like linear regression, we draw random values from the predicted distribution. In the case of linear regression that is: a normal distribution with mean equal to the average prediction and standard deviation equal to the residual standard deviation. This both takes the model into account but also by using the residual standard deviation and drawing random numbers, we ensure that when generating multiple imputed datasets, we don't pretend we can fill in missing data values perfectly. We fill them in with some error / uncertainty, which is important for the MI to generate accurate standard errors, confidence intervals and p-values. For non-parametric models, such as random forest machine learning models, we do not have an assumed distribution so we cannot draw random values from a distribution with some mean and standard deviation estimate. Instead, uncertainty in predictions must be captured in other ways, such as via bootstrapping. However, the details of this are beyond the scope of this lecture. Finally, when

We do not discuss it here, but many of these limitations can be addressed using alternate prediction models, especially in machine learning.

Once multiple imputed datasets are generated, the previously discussed steps are performed. The analysis of interest is run on each dataset to generate estimates for the parameters of interest, $\hat{Q}$ and standard errors and these are combined (pooled) to generate an overall estimate and an uncertainty estimate that accounts for sampling variation, missing data uncertainty, and simulation error from finite number of imputed datasets. These are then reported as usual (estimate, standard error, confidence interval, p-value, etc.).

2.3 *MI in R*

Before we can try MI in R we need to make a dataset with missing values. It is possible to do multilevel imputation, but it is more complicated and outside the scope of this lecture. To begin with, we will make a single level dataset. For the sake of example, we will take only complete data and then add missing data ourselves. This allows us to compare different approaches with the "truth" from the non-missing data.

The code below is not important to understand. It just creates three datasets:

1. The raw daily diary dataset we've worked with
2. A complete case, between person dataset with no missing values.
3. A between person dataset with missing values

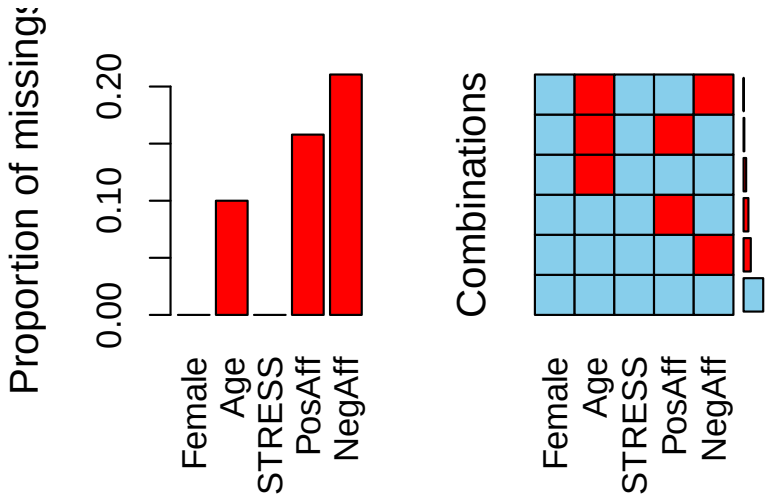
```
## read in the dataset
d <- readRDS("aces_daily_sim_processed.RDS")

## between person data, no missing
davg <- na.omit(d[, .(Female = factor(na.omit(Female)[1],
    levels = 0:1), Age = na.omit(Age)[1], STRESS = mean(STRESS,
    na.rm = TRUE), PosAff = mean(PosAff, na.rm = TRUE),
    NegAff = mean(NegAff, na.rm = TRUE)), by = UserID])

## create missing data
davgmiss <- copy(davg)
davgmiss[STRESS < 1, `:=`(NegAff, NA)]
davgmiss[STRESS > 4, `:=`(PosAff, NA)]
## random missingness on age
set.seed(1234)
davgmiss[, `:=`(Age, ifelse(rbinom(.N, size = 1,
    prob = 0.1) == 1, NA, Age))]
```

```
## drop unneeded variables to make analysis
## easier
davgmiss[, ':='(UserID, NULL)]
```

Before doing any imputation, it is a good idea to examine data. The `VIM` package in `R` has helpful functions for exploring missing data. We start by looking at the amount missing on each variable and the patterns of missing data using the `aggr()` function. This shows us that just over half of cases have complete data on all variables.

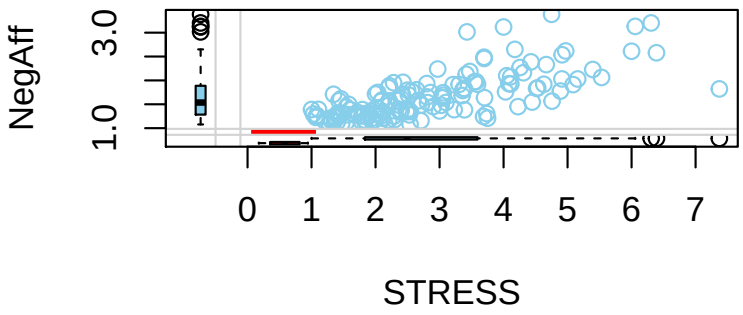
```
aggr(davgmiss, prop = TRUE, numbers = TRUE)
```

```
## Warning in plot.aggr(res, ...): not enough
## horizontal space to display frequencies
```



A margin plot is an augmented scatter plot that in the margins shows the values on one variable when missing on the other along with a boxplot. This shows us that when negative affect is missing, stress scores are all low between 0 and 1. There are no missing stress scores so there is only one boxplot for negative affect.

```
marginplot(davgmiss[, .(STRESS, NegAff)])
```



We can test whether levels of one variable differ by missing on another variable. This is often done with simple analyses, like t-tests or chi-square tests.

```
## does stress differ by missing on negative
## affect?
t.test(STRESS ~ is.na(NegAff), data = davgmiss)
```

```
##
##  Welch Two Sample t-test
##
## data:  STRESS by is.na(NegAff)
## t = 20, df = 200, p-value <2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  2.0 2.5
## sample estimates:
## mean in group FALSE  mean in group TRUE
##                2.83                0.58
```

```
## does stress differ by missing on positive
## affect?
t.test(STRESS ~ is.na(PosAff), data = davgmiss)
```

```
##
##  Welch Two Sample t-test
##
## data:  STRESS by is.na(PosAff)
## t = -20, df = 50, p-value <2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
```

```
##  -3.4 -2.7
## sample estimates:
## mean in group FALSE  mean in group TRUE
##                 1.9                 4.9
```

```
## does age differ by missing on negative
## affect?
t.test(Age ~ is.na(NegAff), data = davgmiss)
```

```
##
##  Welch Two Sample t-test
##
## data:  Age by is.na(NegAff)
## t = -2, df = 60, p-value = 0.02
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -1.82 -0.17
## sample estimates:
## mean in group FALSE  mean in group TRUE
##                  21                  22
```

```
## does age differ by missing on positive
## affect?
t.test(Age ~ is.na(PosAff), data = davgmiss)
```

```
##
##  Welch Two Sample t-test
##
## data:  Age by is.na(PosAff)
## t = 2, df = 40, p-value = 0.08
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.1  1.7
## sample estimates:
## mean in group FALSE  mean in group TRUE
##                  22                  21
```

```
chisq.test(davgmiss$Female, is.na(davgmiss$PosAff))
```

```
##
##  Pearson's Chi-squared test with Yates'
##  continuity correction
##
## data:  davgmiss$Female and is.na(davgmiss$PosAff)
## X-squared = 0.07, df = 1, p-value = 0.8
```

```
chisq.test(davgmiss$Female, is.na(davgmiss$NegAff))
```

```
##
##  Pearson's Chi-squared test with Yates'
##  continuity correction
##
## data:  davgmiss$Female and is.na(davgmiss$NegAff)
## X-squared = 0.4, df = 1, p-value = 0.5
```

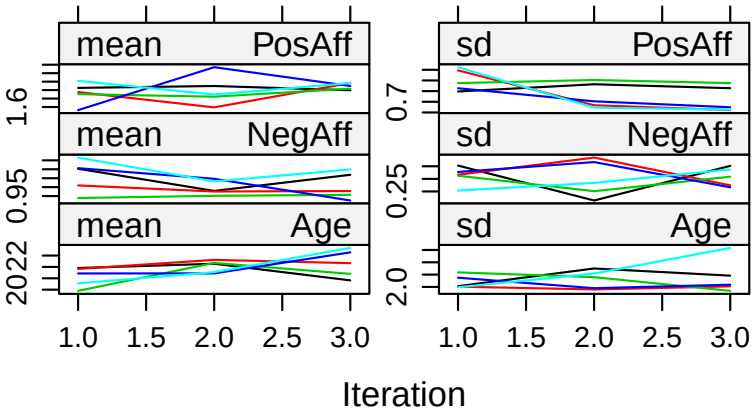
Actually conducting MI in R is relatively straight forward. A very flexible, robust approach is available using the `mice` package and the `mice()` function. You can control almost every aspect of the MI in `mice()` but you also can use defaults.

`mice()` takes a dataset, the number of imputed datasets to generate, the number of iterations, the random seed (make it reproducible) and default imputation methods. The defaults are fine, but you can change default imputation methods. Below we ask for all GLMs:

1. "norm" which is linear regression for numeric data
2. "logreg" which is logistic regression for factor data with 2 levels
3. "polyreg" which is polytomous or multinomial regression for unordered factor data with 3+ levels
4. "polr" ordered logistic regression for factor data with 3+ ordered levels

R will do the steps discussed to then generate imputed datasets. We plot them to see if there is evidence of convergence. From 3 iterations, it may be too few to see evidence of convergence.

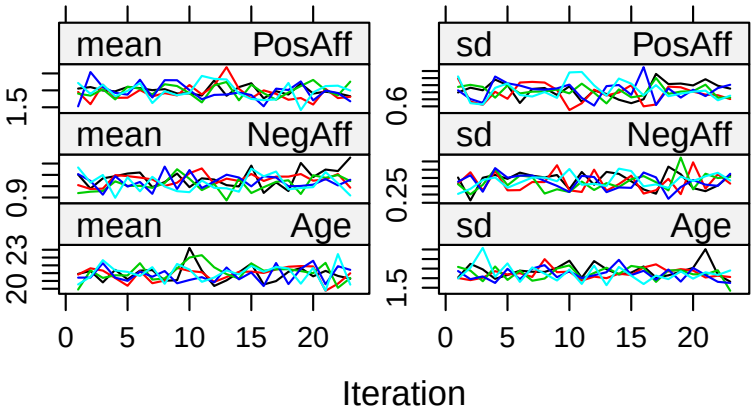
```
mi.1 <- mice(davgmiss, m = 5, maxit = 3, defaultMethod = c("norm",
    "logreg", "polyreg", "polr"), seed = 1234,
    printFlag = FALSE)

## plot convergence diagnostics
plot(mi.1, PosAff + NegAff + Age ~ .it | .ms)
```



If we are not convinced the model converged, re-run with more iterations. Convergence is when there is no clear systematic change with additional iterations.

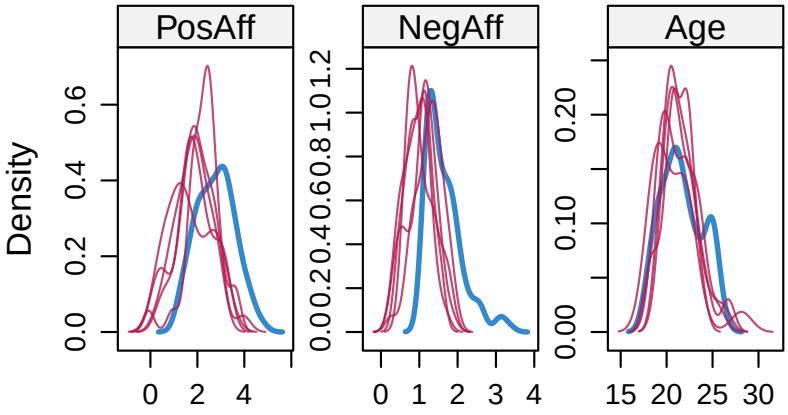
```
## run an additional iterations
mi.1 <- mice.mids(mi.1, maxit = 20, printFlag = FALSE)
```

```
## plot convergence diagnostics
plot(mi.1, PosAff + NegAff + Age ~ .it | .ms)
```

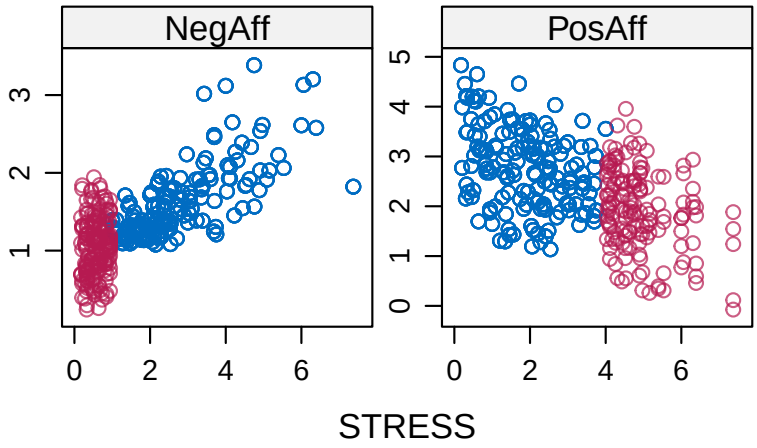


Once we are satisfied with convergence, we can plot the imputed data versus observed data. Imputed data are in red.

```
densityplot(mi.1, ~PosAff + NegAff + Age)
```




```
xyplot(mi.1, NegAff + PosAff ~ STRESS)
```



Once you have created the imputed datasets, you can analyse them. Recall that analyses must be repeated on each individual dataset. For models that are fitted with a formula interface, you can use the special `with()` function which works with multiply imputed datasets. It means that the analysis is automatically run on each imputed dataset separately. If we view the results, we end up with a bunch of separate linear regressions. To pool the results from the individual regressions, we use the `pool()` function[6].

```
mi.reg <- with(mi.1, lm(NegAff ~ PosAff + Age))

mi.reg

## call :
```

[6] The pooled output has several pieces: **estimate**: the pooled, average regression coefficients; **ubar**: the average variance (based on the squared standard errors) associated with sampling variation, we called it $\bar{V}$; **b**: the between imputation variance; **t**: the total variance incorporating sampling variation, between imputation variance, and simulation error; **dfcom**: the degrees of freedom for the complete data analysis (i.e., if there had not been missing data); **df**: the estimated residual degrees of freedom for the model, taking into account missingness; **riv**: the relative increase variance due to missing data; **lambda** or λ: the proportion of total variance due to missing data, that is: $\lambda = \frac{B+B/m}{T}$; **fmi**: the fraction of missing information

```
## with.mids(data = mi.1, expr = lm(NegAff ~ PosAff + Age))
##
## call1 :
## mice.mids(obj = mi.1, maxit = 20, printFlag = FALSE)
##
## nmis :
## Female    Age STRESS PosAff NegAff
##      0     19      0     30     40
##
## analyses :
## [[1]]
##
## Call:
## lm(formula = NegAff ~ PosAff + Age)
##
## Coefficients:
## (Intercept)       PosAff          Age
##      2.4322      -0.1792      -0.0186
##
##
## [[2]]
##
## Call:
## lm(formula = NegAff ~ PosAff + Age)
##
## Coefficients:
## (Intercept)       PosAff          Age
##      2.8535      -0.2324      -0.0342
##
##
## [[3]]
##
## Call:
## lm(formula = NegAff ~ PosAff + Age)
##
## Coefficients:
## (Intercept)       PosAff          Age
##      2.8559      -0.2063      -0.0363
##
##
## [[4]]
##
## Call:
## lm(formula = NegAff ~ PosAff + Age)
```

```
##
## Coefficients:
## (Intercept)       PosAff          Age
##      2.8055      -0.2590      -0.0283
##
##
## [[5]]
##
## Call:
## lm(formula = NegAff ~ PosAff + Age)
##
## Coefficients:
## (Intercept)       PosAff          Age
##      3.1198      -0.2434      -0.0457
```

```
pool(mi.reg)
```

```
## Class: mipo    m = 5
##             estimate    ubar      b       t
## (Intercept)    2.813 0.12403 0.0607 0.19682
## PosAff        -0.224 0.00157 0.0010 0.00277
## Age           -0.033 0.00024 0.0001 0.00036
##             dfcom df  riv lambda  fmi
## (Intercept)   187 23 0.59   0.37 0.42
## PosAff        187 18 0.76   0.43 0.49
## Age           187 28 0.50   0.33 0.38
```

We can also summarise the pooled results with confidence intervals using the `summary()` function on the pooled results[7]. Finally, we can get a pooled R^2 value using the `pool.r.squared()` function on the analyses.

[7] The summary output includes standard regression output in this case. **estimate**: the pooled, average regression coefficients; **std.error**: the standard error of the estimates incorporating all three sources of MI error, specifically: $se = \sqrt{T}$; **statistic**: $\frac{estimate}{std.error}$; **df**: the estimated degrees of freedom incorporating uncertainty due to missing data, combined with the statistic to calculate p-values; **p.value**: the p-value, the probability of obtaining the estimate or larger in the sample given that the true population value was zero. **2.5%**: the lower limit of the 95% confidence interval; **97.5%**: the upper limit of the 95% confidence interval.

```
m.mireg <- summary(pool(mi.reg), conf.int = TRUE)
print(m.mireg)
```

```
##             estimate std.error statistic df
## (Intercept)    2.813     0.444       6.3 23
## PosAff        -0.224     0.053      -4.3 18
## Age           -0.033     0.019      -1.7 28
##             p.value  2.5 %  97.5 %
## (Intercept) 7.3e-07  1.896  3.7303
## PosAff      2.1e-04 -0.335 -0.1133
## Age         9.7e-02 -0.072  0.0064
```

```
pool.r.squared(mi.reg)
```

```
##      est lo 95 hi 95 fmi
## R^2 0.17 0.051  0.32 NaN
```

In general, you can use any analysis you want with multiple imputation, replacing `lm()` with another function, including `lmer()` as we've worked with this semester.

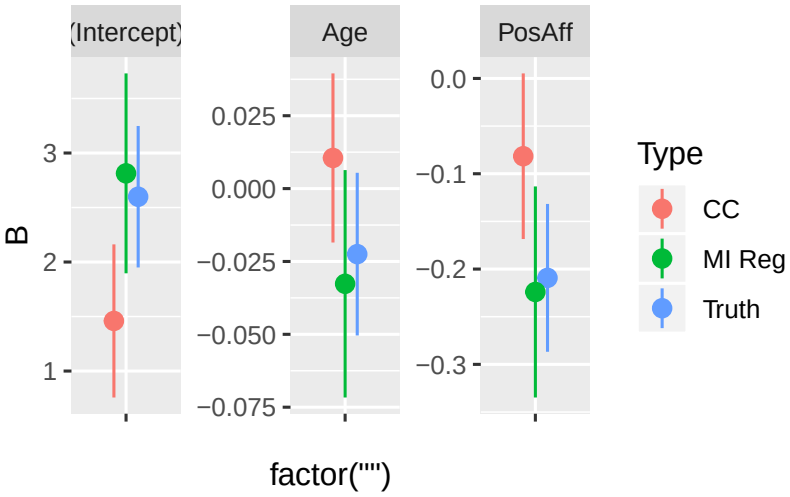
2.4 *Multiple Imputation Comparison*

One thing that often is helpful is to compare results under different models or assumptions. The code that follows is a bit complicated (don't worry, you don't need to use it yourself), and is used to fit the same model in three ways:

1. pooled from the multiple imputations that was already done, labelled "MI Reg".
2. based on the true, non-missing data, since we made the missing data ourselves and have the true data, labelled, "Truth".
3. Complete case analysis using listwise deletion, labelled "CC".

The graph that follows shows that compared to the true data, the multiply imputed results have larger confidence intervals slightly. In this case, the multiply imputed results are closer to the truth in the absolute estimates (dots in the graph) than are the estimates from the complete case analysis.

```
m.true <- lm(NegAff ~ PosAff + Age, data = davg)
m.cc <- lm(NegAff ~ PosAff + Age, data = davgmiss)

res.true <- as.data.table(cbind(coef(m.true),
    confint(m.true)))
res.cc <- as.data.table(cbind(coef(m.cc), confint(m.cc)))
res.mireg <- as.data.table(m.mireg[, c("estimate",
    "2.5 %", "97.5 %")])
setnames(res.true, c("B", "LL", "UL"))
setnames(res.cc, c("B", "LL", "UL"))
setnames(res.mireg, c("B", "LL", "UL"))

res.compare <- rbind(cbind(Type = "Truth", Param = names(coef(m.true)),
    res.true), cbind(Type = "CC", Param = names(coef(m.true)),
    res.cc), cbind(Type = "MI Reg", Param = names(coef(m.true)),
    res.mireg))

ggplot(res.compare, aes(factor(""), y = B, ymin = LL,
    ymax = UL, colour = Type)) + geom_pointrange(position = position_dodge(0.4)) +
    facet_wrap(~Param, scales = "free")
```


(Intercept)
Age
PosAff
B
3
2
1
0.025
0.000
−0.025
−0.050
−0.075
0.0
−0.1
−0.2
−0.3
factor("")
Type
CC
MI Reg
Truth