Adjusting for Measurement Error and Missing Data Using Bayesian Statistics

Jose Pina-Sánchez and Albert Varela

02 January, 2024

Introduction

This is the third practical exercise from day 2 of the short course 'Adjustment Methods for Data Quality Problems: Missing Data, Measurement Error, and Misclassification.' You can download the associated lecture slides and data here: https://github.com/jmpinasanchez/measurement

This workshop is a first approximation to measurement error and missing data adjustments using Bayesian statistics. We will demonstrate the unparalleled flexibility of Bayesian statistics by undertaking a couple of adjustments that we would not be able to carry out using any of the methods we have seen so far. In the first exercise we will see how we can adjust for all the forms of measurement error seen in police data simultaneously. In the second exercise we will see how we can easily account for both missing data and measurement error in our data by including imputation and measurement models.

Downloading and installing `JAGS` and `rjags`

We are going to use the package `rjags` to call the software `JAGS` from `R`. There are other great packages to run Bayesian statistics that are faster (see for example `rstan`), maintained and updated more regularly, or perhaps simpler to use (`blavaan`). However, we chose to work with `rjags` since out of the five packages that we considered (`rjags`, `rstan`, `Brugs`, `brms`, `blavaan`), `rjags` is the one that we find to offer the necessary flexibility to specify the different types of measurement error mechanisms seen in police data (multiplicative, systematic, random, and differential), while remaining relatively intuitive.

First we need to install `JAGS`. You can download the Windows and Mac installer for `JAGS` from https://sourceforge.net/projects/mcmc-jags. Simply download and run the latest installer to install `JAGS` on your machine. In that same website you can also find a short workshop introducing `JAGS` written by Martyn Plumer, which you might find useful if you want to keep learning about `JAGS` and Bayesian statistics.

The second step is to install `rjags` so we can call `JAGS` directly from `R`.

```
install.packages("rjags")
```

We can now proceed to load the package with the `library` command. We also use `set.seed` to ensure reproducibility, and we are ready to go.

```
library(rjags)
set.seed(5)
```

Exercise 1. Accounting for multiple forms of measurement error

As we saw in our first workshop, we cannot retrieve the true values of a variable prone to measurement error by simply simulating the measurement error process. This represents a key limitation of methods like `rcme`, and the adjustment of police recorded crime data in particular, since in addition to underreported crimes (systematic errors), we expect to see unexplained (random) inconsistencies in crime recording between and within police forces. Random errors might not be the most damaging types of errors present in police data,

however, it is unquestionable that not been able to provide a fully accurate description of the measurement error mechanisms present in police data limits the precision of our adjustments.

Furthermore, when the number of recorded crimes in a given area, at a given period, stems from low level geographical units (such as LSOAs or neighbourhoods) and/or short periods of time (such as days or months), chances are that a good share of the area-period observations will be 0s (especially so when we are considering more serious and, therefore, less frequent crimes). When that is the case, we won't be able to retrieve the true extent of crime by simulating the measurement error process. This is because we assumed multiplicative errors, so the simulated error will be made artificially equal to zero in those person-period observations when no crimes were recorded ($0 \cdot U = 0$).

We are going to see how we can overcome both of these limitations using adjustments based on Bayesian statistics. The key idea behind this approach is to estimate simultaneously the outcome model of interest (where we explore the causes or consequences of crime) and the measurement error model (where we describe how well does police data reflect the true extent of crime). This differs from the `rcme` approach, which is based on estimating the measurement error model first, so we can adjust the crime rates accordingly, which are then used in the outcome model in a separate stage.

We will illustrate the implementation of our adjustments using a simple dataset exploring the effect of education on crime. This topic has been studied at great length, especially within the field of Economics of Crime (Groot & van den Brink, 2010; Lochner, 2020). Most studies in the literature point at higher education levels associated with lower crime rates, and a few of them seek to adjust for measurement error using methods like instrumental variables and generalised method of moments (Buonanno & Montolio, 2008; Fajnzylber et al., 2002). These methods can help to partially adjust for the impact of systematic and random errors, however, as far as we are aware, they cannot be used to adjust for differential errors (where the measurement error is associated with the predictors or outcome of interest).

This problem should not be overlooked, since, as demonstrated by Buil-Gil et al. (2020), crime underreporting tends to be more pronounced in areas with low education levels. In this exercise we will see how Bayesian adjustments afford a remarkable degree of flexibility with which to explore the impact of practically any type of measurement error mechanism that we could suspect to be affecting our data, including differential errors.

Importing, cleaning and exploring the data

All the analyses included in this workshop are based on the same data: the number of violent crimes recorded in 2011, across the close to five thousand Lower Super Output Areas (LSOAs) that compose Greater London; which we seek to explain using three covariates derived from the 2011 Census: (i) the percentage of residents with no qualification, our focal variable, and two other controls, (ii) the median house price, and (iii) the number of residents in the area. This dataset, together with more than a hundred other area characteristics from the Census and different crime types from data.police.uk, is available in the *data* section of the Recounting Crime website, under *Census and police data*.

We can download the data or import it directly into R calling it from the github repository of the *Recounting Crime* project. We then proceed to drop all the variables that we will not use in our examples, and simplify the name of the variables we keep.

```
#Importing the data.
data = read.csv(
'https://raw.githubusercontent.com/davidbuilgil/london-database/main/crimes_london_2011.csv')
#Selecting variables.
data = data[c("Mid.year.Population.Estimates.All.Ages.2011",
              "House.Prices.Median.Price.2011", "Qualifications...No.qualifications.2011",
              "VIOLENCE_WITH_INJURY_2011")]
#Renaming variables.
names(data) = c("residents", "price", "noqual", "violence")
```

We can now take a quick look at the four variables considered.

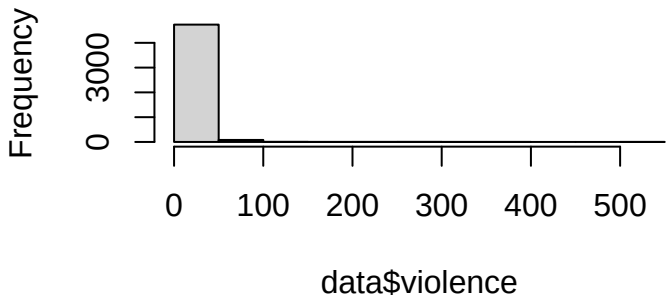
```r
#Exploratory analysis.
head(data)
```

```
##   residents  price noqual violence
## 1      1731 240500     18       10
## 2      1417 212000     15       59
## 3      1569 140000     21       11
## 4      1798 226500     19       40
## 5      2889 151475     12       90
## 6      1655 214000     18        8
```

```r
summary(data)
```

```
##    residents       price              noqual      violence
##  Min.   : 997   Length:4829        Min.   : 1   Min.   :  0
##  1st Qu.:1534   Class :character   1st Qu.:13   1st Qu.:  5
##  Median :1660   Mode  :character   Median :18   Median :  9
##  Mean   :1697                      Mean   :18   Mean   : 13
##  3rd Qu.:1822                      3rd Qu.:23   3rd Qu.: 16
##  Max.   :4973                      Max.   :43   Max.   :546
```

```r
hist(data$violence, main = NULL)
```



Our outcome variable follows a typical count data distribution, left-censored at zero, and in this case heavily right-skewed, including some large outliers. Since in this exercise we are only trying to provide an illustration about how to use Bayesian statistics to adjust for measurement error, and given that strong outliers tend to hinder convergence (more on this below), we will simply drop the 13 LSOAS which recorded more than 100 violent crime counts.

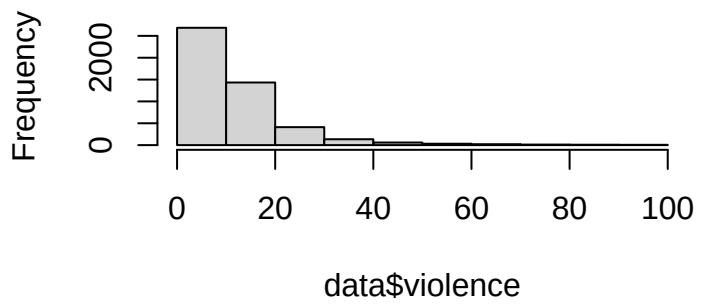
Notice as well how each variable is measured in a different unit. To facilitate model convergence we will redress this by expressing the number of residents in thousands, the median house price in hundreds of thousands (of pounds), and the percentage of residents with no qualification in tens of percentage points. For the same reason of facilitating convergence, but also to facilitate the interpretation of our findings, we center each explanatory variable around their mean. We will also set the median house price as a numeric variable, and remove the 13 missing cases which turned it into a character variable when we imported it.

```r
#Transforming covariates to facilitate convergence.
data$residents = data$residents/1000
data$residents = data$residents - mean(data$residents)
data$price = as.numeric(data$price)/100000
#Dropping 13 cases with missing values for price.
```

```
data = na.omit(data)
data$price = data$price - mean(data$price)
data$noqual = data$noqual/10
data$noqual = data$noqual - mean(data$noqual)
#Dropping outliers.
data = data[which(data$violence<=100),]
#Checking that the transformations were carried out as expected.
summary(data)
```

```
##    residents        price          noqual         violence
##  Min.   :-0.7   Min.   :-2.7   Min.   :-1.68   Min.   :  0
##  1st Qu.:-0.2   1st Qu.:-1.1   1st Qu.:-0.52   1st Qu.:  5
##  Median : 0.0   Median :-0.6   Median :-0.02   Median :  9
##  Mean   : 0.0   Mean   : 0.0   Mean   : 0.00   Mean   : 12
##  3rd Qu.: 0.1   3rd Qu.: 0.4   3rd Qu.: 0.50   3rd Qu.: 16
##  Max.   : 3.3   Max.   :30.5   Max.   : 2.55   Max.   :100
```

```
hist(data$violence, main = NULL)
```



We now proceed to estimate our outcome model, so we can estimate the association between percentage of residents with no qualification and counts of violent crime conditional on number of residents and median house price. To do so we are going to estimate a Poisson model using the `glm` command. We are going to call this model *naive_poisson_freq* because we are not adjusting for measurement error yet, hence *naive*, and because we are using frequentist statistics, hence *freq*.

As a side note, notice how we cannot simply log-transform violent crime and regress it using a linear model since $ln(0)$ is undefined. Other types of transformations are commonly used to 'normalise' right-skewed variables including zeros, such as $\sqrt{X}$ or $ln(X+1)$. These can be useful when the research question seeks to predict, but when we are trying to explain, using such transformations should be discouraged as they hinder interpretability of findings.

```
#Naive model using frequentist statistics.
naive_poisson_freq = glm(violence ~ noqual + price + residents, data=data,
                         family="poisson")
summary(naive_poisson_freq)
```

```
##
## Call:
## glm(formula = violence ~ noqual + price + residents, family = "poisson",
##     data = data)
```

```
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  2.44081    0.00434   561.9   <2e-16 ***
## noqual       0.11024    0.00686    16.1   <2e-16 ***
## price       -0.05514    0.00312   -17.6   <2e-16 ***
## residents    0.91292    0.01186    77.0   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 39920  on 4802  degrees of freedom
## Residual deviance: 33330  on 4799  degrees of freedom
## AIC: 52328
##
## Number of Fisher Scoring iterations: 5
```

The three explanatory variables are significant, and point in the expected directions. Violent crime is negatively associated with the median house price, and positively associated with the number of residents and the percentage of those residents with no qualification. Specifically, for every ten percentage point increase in residents without a qualification we estimate a 12% increase in violent crime, holding the number of residents and house price constant. This figure can be derived by taking the exponential of the regression coefficient for percentage of residents with no qualification, which gives us its rate ratio.

```
#Calculating the rate ratio for the percentage of residents with no qualification.
exp(naive_poisson_freq$coefficients[2])
```

```
## noqual
##    1.1
```

Bayesian modelling in rjags

We start our Bayesian adjustments by replicating the naive Poisson model that we estimated above, only now we use Bayesian statistics. We will go over this Bayesian specification rather quickly, if you want to know more about count models in JAGS we recommend the following tutorial from George Derpanopoulos (2016). To estimate a model in JAGS we need to undertake five steps: 1) specify the model using rjags syntax (including prior distributions); 2) connect the model to the dataset we are using in R; 3) run the MCMC sampler; 4) check that the model has converged; and 5) extract and analyse our results.

The following code can be used to specify our naive Poisson model in JAGS.

```
#Naive Poisson model.
model_string = "
model {
  # Likelihood function
    # Outcome model (Poisson)
  for (i in 1:4803) {
    violence[i] ~ dpois(lambda[i])
    log(lambda[i]) <- beta0 + beta1*noqual[i] + beta2*price[i] + beta3*residents[i]
  }
  # Priors
beta0 ~ dnorm(0, 0.00001)
beta1 ~ dnorm(0, 0.00001)
beta2 ~ dnorm(0, 0.00001)
beta3 ~ dnorm(0, 0.00001)
```

```
}
"
```

Notice how we divide the code in two parts, a likelihood function (i.e. the conditional probability of each of the parameters included in our Poisson model given the data), and prior probabilities for all those parameters (i.e. our prior knowledge about the value of those parameters). Following Bayes theorem these two probabilities are combined to derive the posterior distribution for each of the parameters included in our model (estimated via MCMC algorithms), from which we will draw our inferences.

Under the model we have specified above, we seek to estimate four parameters, three slopes (*beta1*, *beta2* and *beta3*), one for each explanatory variable included in the model, and one intercept (*beta0*). Notice how we use non-informative prior probabilities for each of these parameters (in this case extremely dispersed normal distributions with mean zero). Such non-informative (also known as diffuse) 'priors' are used to reflect that we have no prior knowledge about the value of the parameters to be estimated, which is also the position that we take by default when we use frequentist statistics.

Next we connect the model code we have just created with the data we are using in R. In addition, we determine that we want two different MCMC chains (these chains represent the computational technique used to approximate the posterior distribution of the parameters to be estimated). We use more than one chain to assess whether the chains have converged (i.e. whether the chains have sufficiently explored the parameter space and have reached a stable distribution that approximates the posterior distribution of interest). We also provide sensible initial values for each of the parameters to be estimated to facilitate convergence - by 'sensible' we mean values that could be expected to fall within the parameter space.

```
#Providing sensible initial values.
inits = list(beta0=1, beta1=0.1, beta2=-0.1, beta3=0.1)
#Connecting the model specification with our data.
model = jags.model(textConnection(model_string), data=data, inits=inits, n.chains=2)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 4803
##    Unobserved stochastic nodes: 4
##    Total graph size: 31732
##
## Initializing model
```

At this point we proceed to run the MCMC sampler. Here we have to decide how many iterations of our MCMC chains we want to use and how many we want to discard. The discarding process is defined through the `n.burnin` option. We want to make sure that we discard initial values of the estimation process, where the MCMC chains have not yet converged, and therefore they are not yet sampling from the posterior distribution that we seek to explore.

The sample size with which we estimate our posterior distributions is determined by the number provided in `n.iter`. The bigger this value, the more precisely we will be able to define the posterior distributions, but also the longer the sampling/estimation process will take. For relatively simple generalised linear models like this one, the choice `n.iter = 2000, n.burnin = 500`, works fine.

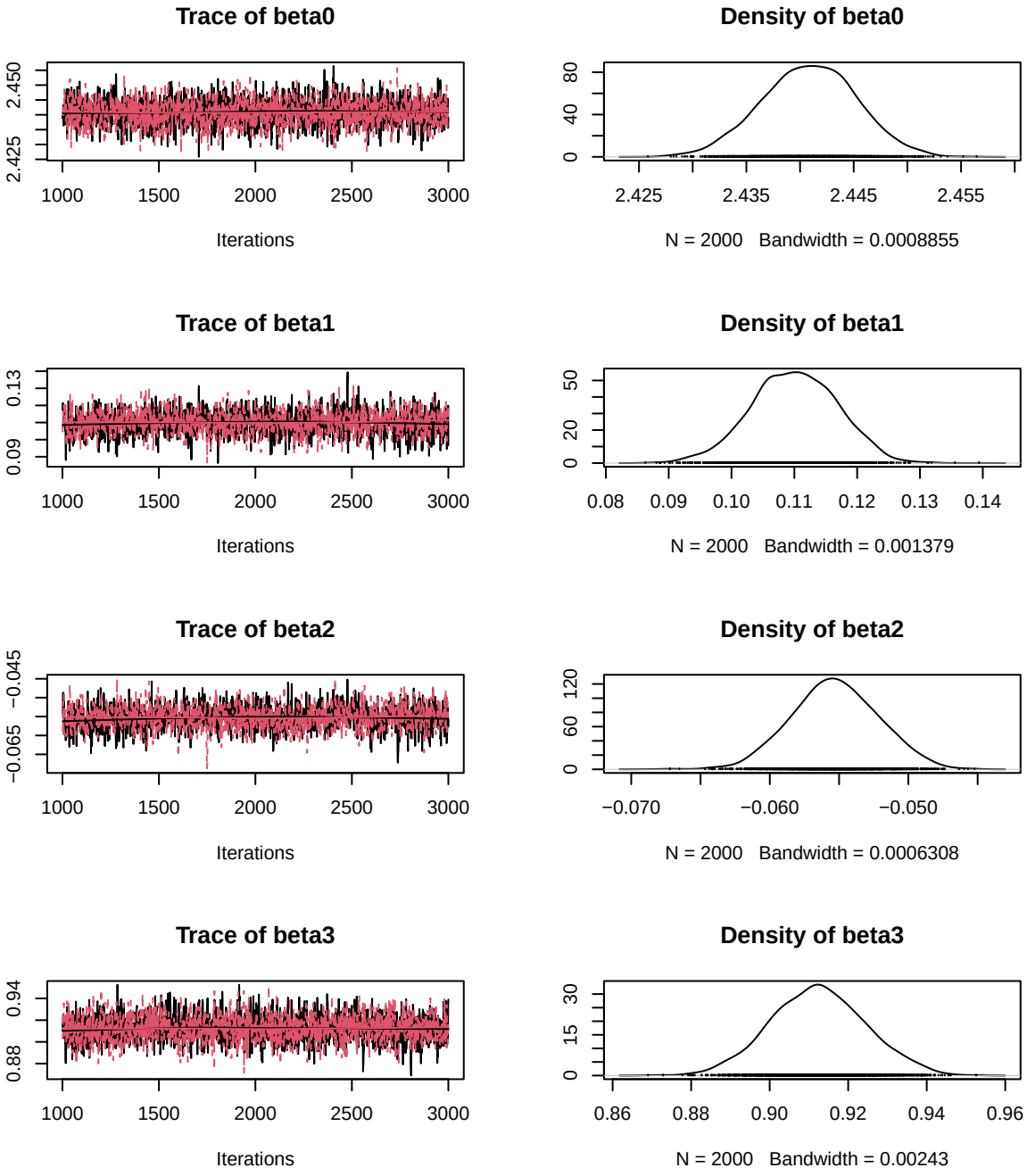
```
#Determining the sample size that we will rely upon to estimate the posterior
#distributions of the parameters included in our model.
samples_naive = coda.samples(model, variable.names = c("beta0", "beta1", "beta2",
                             "beta3"), n.iter=2000, n.burnin=500, n.chains=2)
```

The best way to be sure that we have a sufficient number of iterations, is to check the convergence of the chains we have run. Below we do so visually, by checking that each of the two chains (represented in black

and red) are drawing values from the same range and mixing recurrently, which indicates that both chains are appropriately exploring the same posterior distributions that week to estimate. These posterior distributions estimated using the number of iterations saved are also shown in the output (they are called *Density of...*).

We can also test for convergence formally using Gelman-Rubin Statistic. This statistic provides a numerical value of convergence. It is constructed by comparing the variance between chains to the variance within a chain, so the closer the statistic is to one, the stronger the evidence of convergence.

```
#Checking convergence visually.
plot(samples_naive)
```



```r
#Testing convergence formally.
gelman.diag(samples_naive)
```

```
## Potential scale reduction factors:
##
##       Point est. Upper C.I.
## beta0          1       1.01
## beta1          1       1.00
## beta2          1       1.00
## beta3          1       1.00
##
## Multivariate psrf
##
## 1
```

The above plots and tests show that convergence was achieved. Expanding the number of iterations to be saved would help the precision with which we estimate our parameters, but what we have is good enough.

We can now move on to analyse our results more concretely, for which we can request a `summary` of the posterior distributions of the parameters that we are estimating. The two key statistics to focus on here are the mean and standard deviation of the posterior distribution of each of our four estimates. These can be considered as roughly equivalent to the point estimates of the regression coefficients and their standard errors reported in frequentist models.

```r
#Results from the naive frequentist model.
summary(naive_poisson_freq)$coefficients
```

```
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)    2.441     0.0043     562  0.0e+00
## noqual         0.110     0.0069      16  4.3e-58
## price         -0.055     0.0031     -18  9.5e-70
## residents      0.913     0.0119      77  0.0e+00
```

```r
#Results from the naive Bayesian model.
summary(samples_naive)$statistics
```

```
##         Mean     SD Naive SE Time-series SE
## beta0  2.441 0.0044  6.9e-05        9.4e-05
## beta1  0.110 0.0068  1.1e-04        1.8e-04
## beta2 -0.055 0.0031  5.0e-05        9.1e-05
## beta3  0.912 0.0120  1.9e-04        2.7e-04
```

As we can see the results from this Bayesian model are practically identical (up to the third decimal) to our previous model, which we estimated using frequentist statistics.

Adjusting for measurement error

Now that we have our outcome model estimated using Bayesian statistics, we can proceed to consider different types of measurement error adjustments. We will undertake adjustments accounting for the types of measurement error mechanisms expected to affect police recorded crime rates. As described in Pina-Sánchez et al. (2022), such measurement error is likely to be: multiplicative (i.e. proportional to the true extent of crime), systematic (as often crimes go under-reported), random (as a result of inconsistencies in recording within and between police forces), and differential (recording rates are associated with other variables included in the model).

Let's start by considering the impact of random multiplicative error in our outcome variable. To do so we are going to introduce a measurement error model in the code as follows: `violence[i] ~ dnorm(true_violence[i]*1, tau_U)`. We will also define a value and a transformation: `sigma_U <- 1`

and `tau_U <- (1 / sigma_U)^2`. Lastly, we will change the outcome model so we substitute the variable affected by measurement error for the unobserved true variable: `true_violence[i] ~ dpois(lambda[i])`. All of this is to indicate that we anticipate the presence of measurement error in the outcome variable taking the following form: $Y^* \sim N(Y \cdot 1, 1)$, where Y^* is the count of violent crimes recorded by the police, and Y is the true but unobserved count of violent crimes.

We take the measurement error process to be non-systematic since we indicate that the mean of recorded crimes is equal to the true crime. However, we now account for a degree of random errors in the recording of violence by indicating that the observed violence is the result of the true value of violence plus some normally distributed errors with a standard deviation equal to 1. This value represents an educated guess, whereby we take into account a moderate level of random errors.

Specifically, the value chosen here indicates an expectation that crime recording inconsistencies can be represented as a normal distribution, which for the 95% of areas where crime recording is more consistent, represents a range of no more than -2 to 2 additional crimes per area. This seems to us a wide enough interval considering that the mean violent count in our sample is 12.

Lastly, the `tau_U <- (1 / sigma_U)^2` transformation is included because `JAGS` define dispersion in statistical distributions using *precision* (the inverse of the variance), but we find more intuitive to use the standard deviation instead.

```
#Poisson model accounting for random errors.
model_string = "
model {
  # Likelihood function
    # Measurement error model (random)
  for (i in 1:4803) {
  violence[i] ~ dnorm(true_violence[i]*1, tau_U)
  }
    # Outcome model (Poisson)
  for (i in 1:4803) {
    true_violence[i] ~ dpois(lambda[i])
    log(lambda[i]) <- beta0 + beta1*noqual[i] + beta2*price[i] + beta3*residents[i]
  }
  # Priors
beta0 ~ dnorm(0, 0.00001)
beta1 ~ dnorm(0, 0.00001)
beta2 ~ dnorm(0, 0.00001)
beta3 ~ dnorm(0, 0.00001)
tau_U <- (1 / sigma_U)^2
sigma_U <- 1
}
"
inits = list(beta0=1, beta1=0.1, beta2=-0.1, beta3=0.1)
model = jags.model(textConnection(model_string), data=data, inits=inits, n.chains=2)
samples_random = coda.samples(model, variable.names = c("beta0", "beta1",
                 "beta2", "beta3"), n.iter=2000, n.burnin=500, n.chains=2)
gelman.diag(samples_random)
```

```
#Comparing results from the naive model and the model adjusting for random errors.
summary(samples_naive)$statistics
```

```
##          Mean     SD Naive SE Time-series SE
## beta0   2.441 0.0044  6.9e-05        9.4e-05
## beta1   0.110 0.0068  1.1e-04        1.8e-04
## beta2 -0.055 0.0031  5.0e-05        9.1e-05
## beta3   0.912 0.0120  1.9e-04        2.7e-04
```

```
summary(samples_random)$statistics
```

```
##          Mean     SD Naive SE Time-series SE
## beta0   2.459 0.0045  7.2e-05        1.1e-04
## beta1   0.101 0.0070  1.1e-04        2.0e-04
## beta2 -0.055 0.0032  5.1e-05        9.5e-05
## beta3   0.894 0.0120  1.9e-04        2.9e-04
```

As expected, after accounting for a degree of noise in our outcome variable, the standard deviations of the posterior distributions of the model's estimates are now larger, except for *beta1*. We can also see that the means of the posterior distributions are very similar. The intercept is slightly larger, while the slopes are somehow smaller. Still, the direction and overall interpretation of the regression coefficients does not change. In particular, considering our focal variable, the conditional association between education and crime remains positive and strong. In sum, it seems that some inconsistencies in the recording of crime should not be expected to have a large biasing effect in our estimate of the potential effect of education on crime.

Once we got to this point, it is quite easy to expand our measurement error model by including a systematic error component. Let's assume that we expect that only 50% of cases of violence are reported and successfully recorded by the police. All we have to do is modify the mean of the measurement error model as follows: `violence[i] ~ dnorm(true_violence[i]*0.5, tau_U)`.

Try to do this yourself and assess the impact that such type of measurement error would have by comparing the results from your adjustment against the naive model, as we just did for the case of an adjustment based on random errors.

################## REMOVE THIS FROM THE HANDOUT ##################

```
#Poisson model accounting for random and systematic errors.
model_string = "
model {
  # Likelihood function
    # Measurement error model (random and systematic negative)
  for (i in 1:4803) {
    violence[i] ~ dnorm(true_violence[i]*0.5, tau_U)
  }
    # Outcome model (Poisson)
  for (i in 1:4803) {
    true_violence[i] ~ dpois(lambda[i])
    log(lambda[i]) <- beta0 + beta1*noqual[i] + beta2*price[i] + beta3*residents[i]
  }
  # Priors
beta0 ~ dnorm(0, 0.00001)
beta1 ~ dnorm(0, 0.00001)
beta2 ~ dnorm(0, 0.00001)
beta3 ~ dnorm(0, 0.00001)
tau_U <- (1 / sigma_U)^2
sigma_U <- 1
}
"
inits = list(beta0=1, beta1=0.1, beta2=-0.1, beta3=0.1)
model = jags.model(textConnection(model_string), data=data, inits=inits, n.chains=2)
samples_systematic = coda.samples(model, variable.names = c("beta0", "beta1",
                     "beta2", "beta3"), n.iter=2000, n.burnin=500, n.chains=2)
gelman.diag(samples_systematic)
```

```r
#Comparing results from the naive model and the model adjusting for random and
#systematic errors.
summary(samples_naive)$statistics
```

```
##          Mean     SD Naive SE Time-series SE
## beta0   2.441 0.0044  6.9e-05        9.4e-05
## beta1   0.110 0.0068  1.1e-04        1.8e-04
## beta2 -0.055 0.0031  5.0e-05        9.1e-05
## beta3   0.912 0.0120  1.9e-04        2.7e-04
```

```r
summary(samples_systematic)$statistics
```

```
##          Mean     SD Naive SE Time-series SE
## beta0   3.167 0.0033  5.3e-05        8.2e-05
## beta1   0.094 0.0050  8.0e-05        1.4e-04
## beta2 -0.054 0.0024  3.8e-05        7.7e-05
## beta3   0.880 0.0090  1.4e-04        2.3e-04
```

As anticipated in Pina-Sánchez et al. (2022), the systematic error introduced in the outcome variable does not seem to bias our estimates, besides the expected increase of the intercept. This is because the multiplicative relationship between the error and the true extent of crime turns into an additive relationship when the outcome variable is specified under a generalised linear model that involves a log-link function, like the Poisson model; $log(Y^*) = log(Y \cdot U) = log(Y) + log(U)$. We do see, however, much smaller standard deviations for the posterior distributions; which could result in type-II errors (failing to detect a significant effect) in naive models that do not account for crime underrecording.

########################## UP TO HERE ##########################

We can now explore differential errors, which take place when the measurement error is associated with the predictors or outcome of interest. In the following example we maintain the same assumptions regarding the presence of inconsistencies and average underrecording, but will also assume that the probability of reporting/recording crime is 10% lower in areas where the number of residents with no qualification is ten percentage points above the mean. This is equivalent to assuming a 50% recording rate of violent crimes in LOAS with average proportion of residents with no qualifications (17.8%), and a recording rate of 45% in LSOAs where the proportion of residents with no qualifications is 27.8%.

To reflect these systematic errors we use: `violence[i] ~ dnorm(true_violence[i]*diff.P[i], tau_U)`, where `diff.P[i]` is the individual probability of recording attributed to each LSOA according to their proportion of residents with no qualifications. These probabilities are derived after defining the association between the error term and the percentage of residents with no qualification using a risk ratio of 0.9, represented in the code as `diff.RR <- 0.9`, which we transform into odds ratios, `diff.OR <- (diff.RR*(1 - recording)) / (1 - (diff.RR*recording))`, so we are ultimately able to derive the differential probabilities of recording as follows: `diff.P[i] <- exp(log(recording/(1-recording)) + log(diff.OR)*noqual[1]) / (1+exp(log(recording/(1-recording)) + log(diff.OR)*noqual[i]))`.

This last expression reflects the estimation of probabilities based on a simple logistic model with its intercept taken to be the baseline recording rate, and its slope being the differential error expressed as log-odds. If you want to consider different levels of baseline recording rates and their association with the focal variable, all you need to do is modify the values assigned to `recording` and `diff.RR` in the model syntax.

Lastly, since we are increasing the complexity of the estimation process by including a new measurement model, we proceed to double the number of iterations to be saved, from 2,000 to 4,000.

```r
#Poisson model accounting for random, systematic errors and differential errors.
model_string = "
model {
  # Likelihood  function
    # Measurement error model (random, systematic negative and differential)
```

```
  for (i in 1:4803) {
    violence[i] ~ dnorm(true_violence[i]*diff.P[i], tau_U)
      #Calculating the differential recording rates for each area
    diff.P[i] <- exp(log(recording/(1-recording)) + log(diff.OR)*noqual[1]) /
                 (1+exp(log(recording/(1-recording)) + log(diff.OR)*noqual[i]))
  }
    # Outcome model (Poisson)
  for (i in 1:4803) {
    true_violence[i] ~ dpois(lambda[i])
    log(lambda[i]) <- beta0 + beta1*noqual[i] + beta2*price[i] + beta3*residents[i]
  }
  # Priors
  beta0 ~ dnorm(0, 0.00001)
  beta1 ~ dnorm(0, 0.00001)
  beta2 ~ dnorm(0, 0.00001)
  beta3 ~ dnorm(0, 0.00001)
  tau_U <- (1 / sigma_U)^2
  sigma_U <- 1
  recording <- 0.5
  #Differential error as a risk ratio
  diff.RR <- 0.9
  #Differential error as an odds ratio: OR = (RR * (1 - p)) / (1 - (RR * p))
  diff.OR <- (diff.RR*(1 - recording)) / (1 - (diff.RR*recording))
}
"
inits = list(beta0=1, beta1=0.1, beta2=-0.1, beta3=0.1)
model = jags.model(textConnection(model_string), data=data, inits=inits, n.chains=2)
samples_differential = coda.samples(model, variable.names = c("beta0", "beta1",
                      "beta2", "beta3"), n.iter=4000, n.burnin=500, n.chains=2)
gelman.diag(samples_differential)
```

```
#Comparing results from the naive model and the model adjusting for random,
#systematic and differential errors.
summary(samples_naive)$statistics
```

```
##         Mean     SD Naive SE Time-series SE
## beta0  2.441 0.0044  6.9e-05        9.4e-05
## beta1  0.110 0.0068  1.1e-04        1.8e-04
## beta2 -0.055 0.0031  5.0e-05        9.1e-05
## beta3  0.912 0.0120  1.9e-04        2.7e-04
```

```
summary(samples_differential)$statistics
```

```
##          Mean     SD Naive SE Time-series SE
## beta0  3.1790 0.0032  3.6e-05        5.6e-05
## beta1 -0.0019 0.0052  5.8e-05        1.1e-04
## beta2 -0.0502 0.0024  2.6e-05        5.4e-05
## beta3  0.8777 0.0084  9.4e-05        1.5e-04
```

As for the previous model considering systematic underrecording, the standard deviations are much smaller than in the naive model. The means of the posterior distributions for median house price and number of residents remain relatively similar too. However, the estimate for the percentage of residents without qualifications has changed drastically. We can see how its effect size has become practically zero. Such change in our main estimate of interest, and therefore in our interpretation of the association between education and crime, is quite remarkable, especially if we consider the relatively modest differential errors introduced in our

measurement model.

In sum, for the specific scenarios that we have considered here, with police recorded violent crime counts used as the outcome model of a Poisson model, we can conclude that many of the error mechanisms seen in police data had a relatively negligible impact. The only exception is when the focal variable affects crime recording. In that case, even relatively modest forms of differential errors can lead us to categorically wrong conclusions, as we have seen here for the case of the relationship between education and violent crime.

Unfortunately - as far as we are aware - these types of differential errors have not been considered in the literature. To make it worse, this is not just a problem affecting the literature exploring the relationship between education and crime. There is evidence of a wide range of other area characteristics commonly taken as causes of crime (e.g. income, ethnic composition, economic activity, etc.) that are likely associated with the probability of crime being reported to the police (Brunton-Smith et al., 2022; Buil-Gil et al., 2021), which underscores the need to undertake sensitivity analyses in every study relying on police data.

Exercise 2. Adjusting for missing data and recall errors in the Cyber Security Breaches Survey

We start this exercise importing the simplified version of the 2023 Cyber Security Breaches Survey that we used in the SIMEX workshop.

```
#Importing the data.
cyber = read.csv('cyber.csv')
#Remember to use the address of the folder where you saved the dataset.
```

You might remember that in this simplified version of the Cyber Security Survey we have just five variables describing 1315 firms that were subject to phishing attacks. These five variables represent: i) a broad definition of the company's *sector* (private sector, charity, or education); ii) the *size* of the firm in terms of number of employees; iii) whether they have a *policy* in place to protect against cyber crime; iv) whether the participant considers that new measures are needed to prevent future breaches/attacks (*response*); and v) the number of phishing attacks experienced over the last twelve months (*phishcount*).

```
head(cyber)
```

```
##           sector size policy response phishcount
## 1 private sector    3      1        0          1
## 2 private sector    2      0        0          5
## 3 private sector    4      0        0          1
## 4 private sector    2      0        0          2
## 5 private sector    8      0        0          2
## 6 private sector    3      0        0         20
```

```
table(cyber$sector, useNA="ifany")
```

```
##
##        charity      education private sector
##            353            271            691
```

```
summary(cyber)[,2:5]
```

```
##       size           policy        response      phishcount
##  Min.   :  -97   Min.   :0.00   Min.   :0.00   Min.   :  -99
##  1st Qu.:    5   1st Qu.:0.00   1st Qu.:0.00   1st Qu.:    2
##  Median :   40   Median :1.00   Median :0.00   Median :    6
##  Mean   :  490   Mean   :0.62   Mean   :0.26   Mean   :  603
##  3rd Qu.:  200   3rd Qu.:1.00   3rd Qu.:1.00   3rd Qu.:   35
##  Max.   :90000   Max.   :1.00   Max.   :1.00   Max.   :100000
```

We noticed negative cases in both *size* and *phishcount*, reflecting instances where participants declined to respond or indicated that they did not know the answer. In our previous workshop we ignored these missing cases using listwise deletion. Here we will see how we can impute them instead using Bayesian statistics. Furthermore, we will see how we can undertake such missing data imputation while simultaneously adjusting for recall errors in reports of phishing attacks experienced in the last twelve months. We will undertake these two adjustments sequentially, but before we do so we need to undertake a few more data cleaning procedures.

```
#Setting non-responses as missing data.
cyber$size = ifelse(cyber$size<0, NA, cyber$size)
cyber$phishcount = ifelse(cyber$phishcount<0, NA, cyber$phishcount)
```

After setting missing cases in *size* and *phishcount* as `NA`, we log-transform these two variables so they are not so heavily right-skewed.

```
#Log-transforming two heavily right-skewed variables.
cyber$logsize = log(cyber$size)
cyber$logphish = log(cyber$phishcount)
```

Lastly, we manually create dummy variables out of the categorical variable *sector*, as this is the format required in `rjgas`.

```
#Turning sector into dummy variables.
cyber$sector_edu = ifelse(cyber$sector=="education", 1, 0)
cyber$sector_pri = ifelse(cyber$sector=="private sector", 1, 0)
#Dropping redundant variables.
cyber$sector = cyber$size = cyber$phishcount = NULL
```

To frame our adjustments we will explore again our first research question from the SIMEX workshop: *Which types of firms tend to experience more phishing attacks?* To do so we will start with a *naive* model where we assume perfectly measured variables and missing data completely at random.

```
naive_freq = lm(logphish ~ logsize + sector_edu + sector_pri + policy, data=cyber)
summary(naive_freq)
```

```
##
## Call:
## lm(formula = logphish ~ logsize + sector_edu + sector_pri + policy,
##     data = cyber)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -3.450 -1.478 -0.237  1.155  9.105
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.2628     0.1732    7.29  5.9e-13 ***
## logsize       0.1900     0.0318    5.98  3.0e-09 ***
## sector_edu    0.0508     0.1738    0.29  0.77005
## sector_pri    0.4942     0.1420    3.48  0.00052 ***
## policy        0.3455     0.1277    2.71  0.00693 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.9 on 1116 degrees of freedom
##   (194 observations deleted due to missingness)
## Multiple R-squared:  0.0543, Adjusted R-squared:  0.0509
## F-statistic:   16 on 4 and 1116 DF,  p-value: 9.49e-13
```

Notice how the above model discarded 194 observations due to missingness. This should not have a huge impact since it is only 15% of the sample, however, we will see how if we estimate the above model using Bayesian statistics we can easily impute these missing cases and in so doing improve the robustness of our analysis. Let's start by replicating the frequentist linear model we just run using Bayesian statistics. To do so, we will create a new dataset where we delete all missing cases, as the `lm` command did automatically.

```
#Listwise deletion.
cyber_lwd = na.omit(cyber)
```

The code that we will use to specify this naive linear model is very similar to that for the naive Poisson model that we estimated in the previous exercise. There are only three differences: i) we use `dnorm` instead of `dpois` to specify that the outcome variable follows a normal distribution, instead of a Poisson distribution; ii) `tau_e ~ dunif(0,100)`, we include a precision term to capture the dispersion of that normal distribution; we use a diffuse prior to indicate that we do not know anything about that parameter; we did not have to do this before because the variance of a Poisson distribution is defined by its mean; iii) and we remove any link functions on `mu[i]` since this is a linear model.

```
#Naive model - assuming missing completely at random and perfect measurements.
model_string = "
model {
  # Likelihood
    # Outcome model (Linear)
  for (i in 1:1121) {
    logphish[i] ~ dnorm(mu[i], tau_e)
    mu[i] <- beta0 + beta1*logsize[i] + beta2*sector_edu[i] + beta3*sector_pri[i] +
              beta4*policy[i]
  }
  # Priors
  beta0 ~ dnorm(0, 0.00001)
  beta1 ~ dnorm(0, 0.00001)
  beta2 ~ dnorm(0, 0.00001)
  beta3 ~ dnorm(0, 0.00001)
  beta4 ~ dnorm(0, 0.00001)
  tau_e ~ dunif(0,100)
}
"
inits = list(beta0=1, beta1=0.2, beta2=0.1, beta3=0.5, beta4=0.3)
model = jags.model(textConnection(model_string), data=cyber_lwd, inits=inits, n.chains=2)
samples_naive = coda.samples(model, variable.names = c("beta0", "beta1", "beta2",
                "beta3", "beta4"),
                n.iter=2000, n.burnin=500, n.chains=2)
```

```
#Checking convergence.
gelman.diag(samples_naive)
```

```
## Potential scale reduction factors:
##
##       Point est. Upper C.I.
## beta0       1.01       1.01
## beta1       1.01       1.03
## beta2       1.00       1.00
## beta3       1.00       1.01
## beta4       1.00       1.01
##
## Multivariate psrf
##
```

```
## 1.01
```

We can see how 2000 iterations is enough to achieve convergence and estimate the posterior distributions precisely.

```
#Comparing results from the frequentist and Bayesian naive models.
summary(naive_freq)$coefficients
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    1.263      0.173    7.29  5.9e-13
## logsize        0.190      0.032    5.98  3.0e-09
## sector_edu     0.051      0.174    0.29  7.7e-01
## sector_pri     0.494      0.142    3.48  5.2e-04
## policy         0.346      0.128    2.71  6.9e-03
```

```
summary(samples_naive)$statistics
```

```
##        Mean    SD Naive SE Time-series SE
## beta0 1.248 0.174  0.00274         0.0112
## beta1 0.191 0.032  0.00051         0.0016
## beta2 0.058 0.174  0.00274         0.0050
## beta3 0.501 0.141  0.00223         0.0065
## beta4 0.353 0.125  0.00197         0.0043
```

We can also demonstrate that results from our two naive models, using frequentist and Bayesian statistics, are practically identical. At this point, once we have specified the outcome model that we seek to explore using Bayesian statistics, we can expand it so it does not rely on the naive assumptions we have considered. We start by considering imputations for *logsize* and *logphish*. To do so we return to the complete dataset, *cyber*.

Adjusting for missing data

We will follow two imputation procedures. For the outcome variable *logphish* we will impute its missing values simply according to the variables included in the outcome model. That is, we will assume missing at random after conditioning on *logsize*, *sector* and *policy*. For the explanatory variable *logsize* we will specify its own imputation model. We will impute its missing values using a linear model `logsize[i] ~ dnorm(mu1[i], tau_e2)`, and three auxiliary variables `mu1[i] <- theta0 + theta1*sector_edu[i] + theta2*sector_pri[i] + theta3*policy[i]`.

```
#Model assuming missing at random and perfect measurements.
model_string = "
model {
  # Likelihood
    # Imputation model
  for (i in 1:1315) {
    logsize[i] ~ dnorm(mu1[i], tau_e1)
    mu1[i] <- theta0 + theta1*sector_edu[i] + theta2*sector_pri[i] + theta3*policy[i]
  }
    # Outcome model (Linear)
  for (i in 1:1315) {
    logphish[i] ~ dnorm(mu2[i], tau_e2)
    mu2[i] <- beta0 + beta1*logsize[i] + beta2*sector_edu[i] + beta3*sector_pri[i] +
              beta4*policy[i]
  }
  # Priors
  beta0 ~ dnorm(0, 0.00001)
  beta1 ~ dnorm(0, 0.00001)
  beta2 ~ dnorm(0, 0.00001)
```

```
  beta3 ~ dnorm(0, 0.00001)
  beta4 ~ dnorm(0, 0.00001)
  theta0 ~ dnorm(0, 0.00001)
  theta1 ~ dnorm(0, 0.00001)
  theta2 ~ dnorm(0, 0.00001)
  theta3 ~ dnorm(0, 0.00001)
  tau_e1 ~ dunif(0,100)
  tau_e2 ~ dunif(0,100)
}
"
inits = list(beta0=1, beta1=0.2, beta2=0.1, beta3=0.5, beta4=0.3,
             theta0=-1, theta1=1, theta2=-1, theta3=1)
model = jags.model(textConnection(model_string), data=cyber, inits=inits, n.chains=2)
samples_imp = coda.samples(model, variable.names = c("beta0", "beta1", "beta2",
                 "beta3", "beta4", "theta0", "theta1", "theta2", "theta3"),
                 n.iter=2000, n.burnin=500, n.chains=2)
gelman.diag(samples_imp)
```

```
#Comparing results between the naive model and the model assuming MAR.
summary(samples_naive)$statistics
```

```
##        Mean    SD Naive SE Time-series SE
## beta0 1.248 0.174  0.00274         0.0112
## beta1 0.191 0.032  0.00051         0.0016
## beta2 0.058 0.174  0.00274         0.0050
## beta3 0.501 0.141  0.00223         0.0065
## beta4 0.353 0.125  0.00197         0.0043
```

```
summary(samples_imp)$statistics
```

```
##          Mean    SD Naive SE Time-series SE
## beta0   1.234 0.167   0.0026         0.0103
## beta1   0.195 0.031   0.0005         0.0017
## beta2   0.091 0.166   0.0026         0.0047
## beta3   0.507 0.134   0.0021         0.0060
## beta4   0.345 0.123   0.0019         0.0045
## theta0  3.272 0.133   0.0021         0.0067
## theta1  0.939 0.158   0.0025         0.0050
## theta2 -0.927 0.130   0.0021         0.0060
## theta3  1.425 0.113   0.0018         0.0041
```

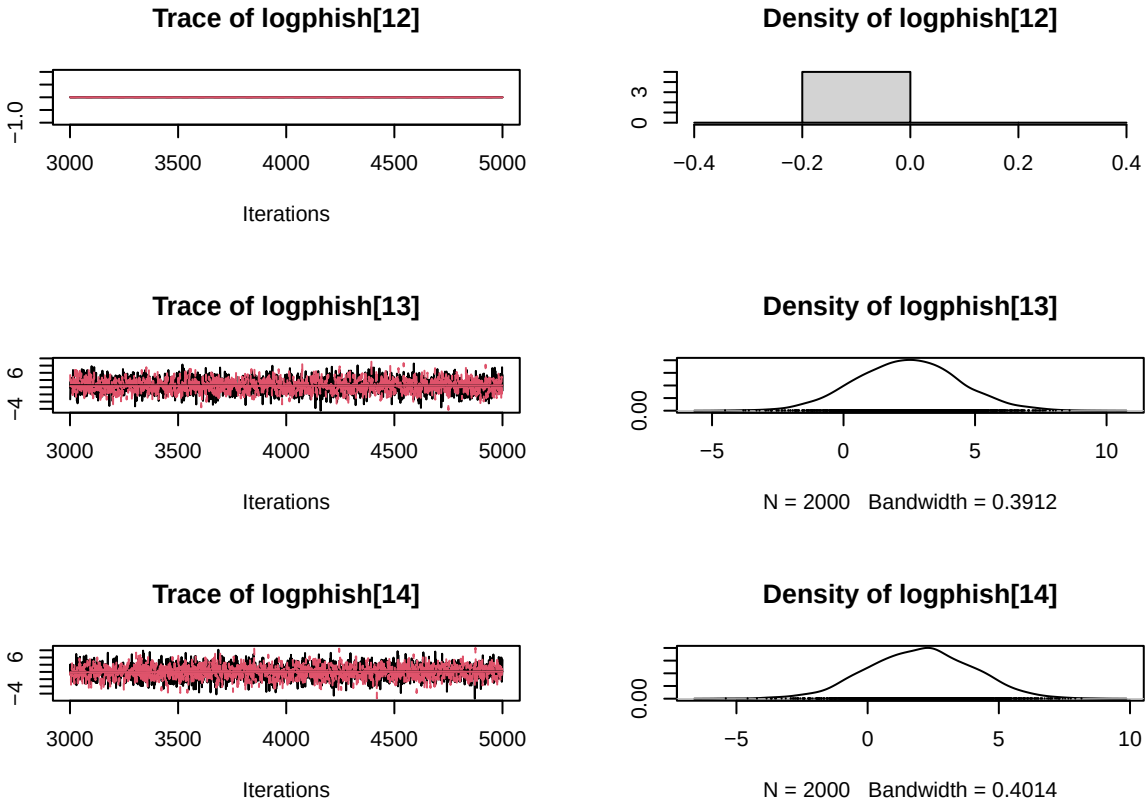
We can see that the imputation process made little difference in the estimates of our outcome model, suggesting that in this case listwise deletion and the assumption of missing completely at random are both valid. This is not to say that the imputation process was entirely useless. All the regression coefficients in the imputation model for *logsize* (*theta0* to *theta4*) were significant. This means that the information available in our auxiliary data helped us predict the missing cases of *logsize*, and in so doing increase the precision of our model.

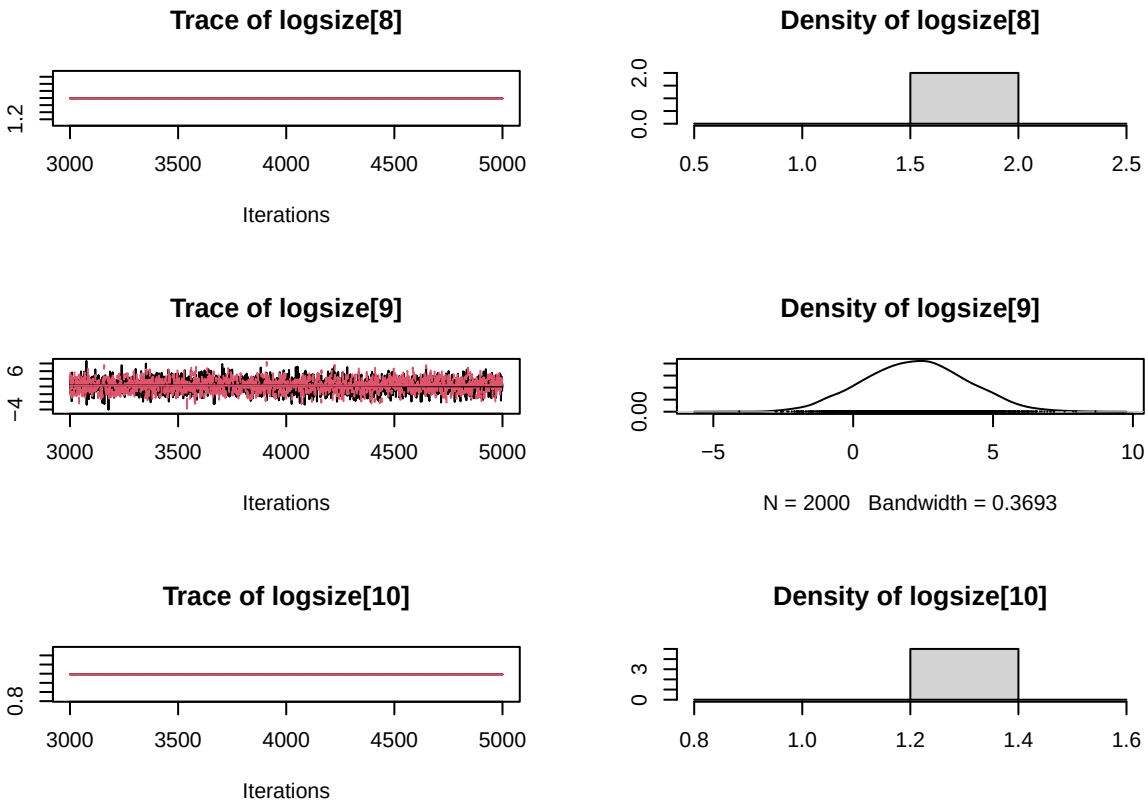
Under a Bayesian approach, we are taking these imputed missing cases as additional model parameters to be estimated, each one with their own posterior distribution, reflecting the precision of our predictions, and transposing the uncertainty associated with this imputation process into the outcome model, without having to undertake multiple imputations. To see this we can obtain a new sample of 2000 iterations but this time requesting to save a few of the values of *logphish* and *logsize*. Specifically, we will save cases 12 to 14 from *logphish* and 8 to 10 from *logsize*. This is so you can appreciate a range of missing and observed cases, but we could potentially save them all, which could be useful to further evaluate the effectiveness of the imputation process.

```r
#Saving the MCMC chains for a few cases of logphish and logsize.
samples_imp = coda.samples(model, variable.names = c("logphish[12:14]",
                        "logsize[8:10]"), n.iter=2000, n.burnin=500, n.chains=2)
summary(samples_imp)$statistics
```

```
##              Mean  SD Naive SE Time-series SE
## logphish[12]  0.0 0.0    0.000          0.000
## logphish[13]  2.4 1.9    0.031          0.034
## logphish[14]  2.0 2.0    0.031          0.031
## logsize[8]    1.8 0.0    0.000          0.000
## logsize[9]    2.2 1.8    0.029          0.028
## logsize[10]   1.4 0.0    0.000          0.000
```

```r
plot(samples_imp)
```





Notice how *logphish* and *logsize* values that were observed in the dataset are taken as given, whereas missing values are estimated. The uncertainty associated to these predictions can be derived from the variability in their posterior distribution.

We could take these imputation models forward and possibly improve their precision further by including additional auxiliary data predictive of the missing values. We left two variables out of the imputation model for *logsize*. These are: *logphish* and *response*. The former could not be used in the imputation model for *logsize* as that would lead to a problem of circularity (*logsize* is already used to predict *logphish* in the outcome model). However, we could expand the imputation model including *response* as an additional auxiliary variable. See if you can do so by expanding our last imputation model, and check whether the inclusion of *response* improves predictions of missing cases of *logsize*.

################## REMOVE THIS FROM THE HANDOUT ##################

```
#Model assuming missing at random including 'response'.
model_string = "
model {
  # Likelihood
    # Imputation model
  for (i in 1:1315) {
    logsize[i] ~ dnorm(mu1[i], tau_e1)
    mu1[i] <- theta0 + theta1*sector_edu[i] + theta2*sector_pri[i] + theta3*policy[i] +
              theta4*response[i]
  }
    # Outcome model (Linear)
  for (i in 1:1315) {
    logphish[i] ~ dnorm(mu2[i], tau_e2)
    mu2[i] <- beta0 + beta1*logsize[i] + beta2*sector_edu[i] + beta3*sector_pri[i] +
```

```
              beta4*policy[i]
  }
  # Priors
  beta0 ~ dnorm(0, 0.00001)
  beta1 ~ dnorm(0, 0.00001)
  beta2 ~ dnorm(0, 0.00001)
  beta3 ~ dnorm(0, 0.00001)
  beta4 ~ dnorm(0, 0.00001)
  theta0 ~ dnorm(0, 0.00001)
  theta1 ~ dnorm(0, 0.00001)
  theta2 ~ dnorm(0, 0.00001)
  theta3 ~ dnorm(0, 0.00001)
  theta4 ~ dnorm(0, 0.00001)
  tau_e1 ~ dunif(0,100)
  tau_e2 ~ dunif(0,100)
}
"
inits = list(beta0=1, beta1=0.2, beta2=0.1, beta3=0.5, beta4=0.3,
             theta0=-1, theta1=1, theta2=-1, theta3=1, theta4=-1)
model = jags.model(textConnection(model_string), data=cyber, inits=inits, n.chains=2)
samples_imp2 = coda.samples(model, variable.names = c("beta0", "beta1", "beta2",
                "beta3", "beta4", "theta0", "theta1", "theta2", "theta3", "theta4"),
                n.iter=2000, n.burnin=500, n.chains=2)
gelman.diag(samples_imp2)
```

```
summary(samples_imp2)$statistics
```

```
##          Mean    SD Naive SE Time-series SE
## beta0   1.243 0.180  0.00285         0.0115
## beta1   0.194 0.033  0.00053         0.0019
## beta2   0.083 0.170  0.00269         0.0051
## beta3   0.504 0.141  0.00224         0.0068
## beta4   0.343 0.126  0.00200         0.0047
## theta0  3.142 0.122  0.00194         0.0057
## theta1  0.921 0.153  0.00241         0.0049
## theta2 -0.906 0.124  0.00195         0.0050
## theta3  1.356 0.109  0.00172         0.0037
## theta4  0.625 0.120  0.00190         0.0026
```

It seems that including *response* improves the imputation model. Specifically, we can see how conditioning on all other auxiliary variables used, firms which are considering responding to future phishing attacks are more likely to be larger in size.

########################## UP TO HERE ##########################

Adjusting for missing data *and* measurement error

At this point we can expand our latest model so it can account for measurement error as well as missing data.

In the SIMEX workshop we discussed how we expect the variable *logphish* to be heavily affected by recall errors, since most managers won't be able to remember the exact number of phishing attacks their firms experienced. We theorised that these errors might take the form of multiplicative random errors, which would be transformed into classical errors if the variable capturing the number of phishing attacks is log-transformed. We also considered that as a result of these recall errors the reliability of *logphish* might be closer to 80% than to 100%, which would be equivalent to the error term having a standard deviation equal to 0.9.

We can use all that information to estimate a measurement model as we simultaneously estimate the imputation and outcome models that we had before.

```
model_string = "
model {
  # Likelihood
    # Measurement error model
  for (i in 1:1315) {
    logphish[i] ~ dnorm(true_logphish[i], tau_U)
  }
    # Imputation model
  for (i in 1:1315) {
    logsize[i] ~ dnorm(mu1[i], tau_e1)
    mu1[i] <- theta0 + theta1*sector_edu[i] + theta2*sector_pri[i] + theta3*policy[i]
  }
    # Outcome model (Linear)
  for (i in 1:1315) {
    true_logphish[i] ~ dnorm(mu2[i], tau_e2)
    mu2[i] <- beta0 + beta1*logsize[i] + beta2*sector_edu[i] + beta3*sector_pri[i] +
    beta4*policy[i]
  }
  # Priors
  beta0 ~ dnorm(0, 0.00001)
  beta1 ~ dnorm(0, 0.00001)
  beta2 ~ dnorm(0, 0.00001)
  beta3 ~ dnorm(0, 0.00001)
  beta4 ~ dnorm(0, 0.00001)
  theta0 ~ dnorm(0, 0.00001)
  theta1 ~ dnorm(0, 0.00001)
  theta2 ~ dnorm(0, 0.00001)
  theta3 ~ dnorm(0, 0.00001)
  tau_U <- (1 / sigma_U)^2
  sigma_U <- 0.9
  tau_e1 ~ dunif(0,100)
  tau_e2 ~ dunif(0,100)
}
"
inits = list(beta0=1, beta1=0.2, beta2=0.1, beta3=0.5, beta4=0.3)
model = jags.model(textConnection(model_string), data=cyber, inits=inits, n.chains=2)
samples_random = coda.samples(model, variable.names = c("beta0", "beta1", "beta2",
                 "beta3", "beta4"),
                 n.iter=2000, n.burnin=500, n.chains=2)
gelman.diag(samples_random)
```

```
#Comparing results between the naive model and the model accounting for both
#missing data and measurement error.
summary(samples_naive)$statistics
```

```
##        Mean    SD Naive SE Time-series SE
## beta0 1.248 0.174  0.00274         0.0112
## beta1 0.191 0.032  0.00051         0.0016
## beta2 0.058 0.174  0.00274         0.0050
## beta3 0.501 0.141  0.00223         0.0065
## beta4 0.353 0.125  0.00197         0.0043
```

```
summary(samples_random)$statistics
```

```
##        Mean    SD Naive SE Time-series SE
## beta0 1.261 0.171  0.00270         0.0135
## beta1 0.191 0.033  0.00051         0.0020
## beta2 0.075 0.165  0.00261         0.0059
## beta3 0.488 0.138  0.00218         0.0078
## beta4 0.349 0.128  0.00202         0.0052
```

We do not see substantial differences after accounting for measurement error in *logphsish*. This is to be expected since the errors are random additive (classical), affecting the outcome variable, and its reliability is considerably high.

We could continue expanding the robustness of our model further by considering adjustments of additional variables prone to measurement error. In the SIMEX workshops we also considered that *logsize* is affected by similar recall errors. We argued that the variable could be approximately measured with 90% reliability, which would mean a standard deviation of the measurement error term equal to 0.7. With that information, would you be able to expand our previous model including an additional measurement model for *logsize*? Do you appreciate any noticeable impact?

################## REMOVE THIS FROM THE HANDOUT ##################

```
model_string = "
model {
  # Likelihood
    # Measurement error model
  for (i in 1:1315) {
    logphish[i] ~ dnorm(true_logphish[i], tau_U1)
  }
  for (i in 1:1315) {
    logsize[i] ~ dnorm(true_logsize[i], tau_U2)
  }
    # Imputation model
  for (i in 1:1315) {
    true_logsize[i] ~ dnorm(mu1[i], tau_e1)
    mu1[i] <- theta0 + theta1*sector_edu[i] + theta2*sector_pri[i] + theta3*policy[i]
  }
    # Outcome model (Linear)
  for (i in 1:1315) {
    true_logphish[i] ~ dnorm(mu2[i], tau_e2)
    mu2[i] <- beta0 + beta1*true_logsize[i] + beta2*sector_edu[i] + beta3*sector_pri[i] +
    beta4*policy[i]
  }
  # Priors
  beta0 ~ dnorm(0, 0.00001)
  beta1 ~ dnorm(0, 0.00001)
  beta2 ~ dnorm(0, 0.00001)
  beta3 ~ dnorm(0, 0.00001)
  beta4 ~ dnorm(0, 0.00001)
  theta0 ~ dnorm(0, 0.00001)
  theta1 ~ dnorm(0, 0.00001)
  theta2 ~ dnorm(0, 0.00001)
  theta3 ~ dnorm(0, 0.00001)
  tau_U1 <- (1 / sigma_U1)^2
  sigma_U1 <- 0.9
  tau_U2 <- (1 / sigma_U2)^2
```

```
  sigma_U2 <- 0.7
  tau_e1 ~ dunif(0,100)
  tau_e2 ~ dunif(0,100)
}
"
inits = list(beta0=1, beta1=0.2, beta2=0.1, beta3=0.5, beta4=0.3)
model = jags.model(textConnection(model_string), data=cyber, inits=inits, n.chains=2)
samples_random = coda.samples(model, variable.names = c("beta0", "beta1", "beta2",
                 "beta3", "beta4"),
                 n.iter=2000, n.burnin=500, n.chains=2)
gelman.diag(samples_random)
```

```
#Comparing results between the naive model and the model accounting for both
#missing data and measurement error in logsize and logphish.
summary(samples_naive)$statistics
```

```
##        Mean    SD Naive SE Time-series SE
## beta0 1.248 0.174  0.00274         0.0112
## beta1 0.191 0.032  0.00051         0.0016
## beta2 0.058 0.174  0.00274         0.0050
## beta3 0.501 0.141  0.00223         0.0065
## beta4 0.353 0.125  0.00197         0.0043
```

```
summary(samples_random)$statistics
```

```
##        Mean    SD Naive SE Time-series SE
## beta0 1.159 0.176  0.00279         0.0123
## beta1 0.222 0.035  0.00056         0.0019
## beta2 0.047 0.170  0.00269         0.0062
## beta3 0.517 0.140  0.00222         0.0078
## beta4 0.306 0.128  0.00202         0.0055
```

We can now see how the regression coefficient for *logsize* has been attenuated in the naive model as a result of the random errors affecting that variable.

########################## UP TO HERE ##########################

Discussion

We conclude this tutorial by underscoring once again the high flexibility afforded by Bayesian statistics. Beyond their identifiability, there are no limits to the type of outcome and measurement error models that can be estimated. Furthermore, we have also seen how we can expand these models to account for missing data, but also account for virtually any other questionable assumption that we suspect might be biasing our estimates. Here we have only scratched the surface.

It is also worth considering how in the above exercises we have provided specific values for the systematic, random or differential part of the error, but a more realistic approach would be to use ranges of values, or even better probability distributions, to reflect the uncertainty surrounding those values. This could be done by using prior distributions for the recording rate, or the standard deviation of the measurement error. To learn more about measurement error and misclassification adjustments using Bayesian statistics we highly recommend a classic textbook on the subject, Gustafson (2003).

References

Buil-Gil, D., Medina, J., and Schlomo, N. (2020). 'Measuring the dark figure of crime in geographic areas: Small area estimation from the Crime Survey for England and Wales'. *The British Journal of Criminology*, 61(2), 364–388.

Buonanno, P., and Montolio, D. (2008). Identifying the socio-economic and demographic determinants of crime across Spanish provinces. *International Review of Law and Economics*, 28(2), 89-97.

Fajnzylber, P., Lederman, D., and Loayza, N. (2002). What causes violent crime?. *European economic review*, 46(7), 1323-1357.

Gallop, M., and Weschle, S. (2019). 'Assessing the impact of non-random measurement error on inference: a sensitivity analysis approach'. *Political Science Research and Methods*, 7(2), 367-384.

Groot, W., and van den Brink, H. M. (2010). 'The effects of education on crime'. *Applied Economics*, 42(3), 279-289.

Gustafson, P. (2003). *Measurement error and misclassification in statistics and epidemiology: Impacts and Bayesian adjustments*. CRC Press.

Lochner, L. (2020). *Education and crime*. In The Economics of Education (pp. 109-117). Academic Press.

Pina-Sánchez, J., Buil-Gil, D., Brunton-Smith, I., and Cernat, A. (2022). 'The impact of measurement error in models using police recorded crime rates'. *Journal of Quantitative Criminology*, 39, 975–1002

Pina-Sánchez, J., Brunton-Smith, I., Buil-Gil, D., and Cernat, A. (2023). 'Exploring the impact of measurement error in police recorded crime rates through sensitivity analysis'. *Crime Science*

Pina-Sánchez, J., Koskinen, J., and Plewis, I. (2019). 'Adjusting for measurement error in retrospectively reported work histories: An analysis using Swedish register data'. *Journal of Official Statistics*, 35(1), 203-229.