Data Generation and Debugging

Sunthud Pornprasertmanit

June 22, 2015

In the previous lecture, we discussed how to use various functions in the apply family to run repeated jobs. The repeated jobs can be implemented by parallel processing using the `parallel` package. In this lecture, data generation for many statistical models is introduced. We will also cover how to generate missing observations. Finally, we will discuss how to debug the written functions that do not work properly.

1 Data Generation

1.1 Multiple Regression

This section will cover the examples of data generation from various models. Let's start with data generation for one of the most basic models, multiple regression.

$$Y = \alpha + \beta_1 X_1 + \cdots + \beta_p X_p + e, \tag{1}$$

where p is the number of predictors, α (sometimes denoted as β_0) is the model intercept, β_p is the regression coefficients of predictor p.

Note that the regression model is a general model that also covers t-test or analysis of variance. For independent t-test, X is a dummy variable that has only two possible values: 0 or 1. For dependent t-test, X and Y are the variables from each group. The difference between the means of X and Y can be tested from the intercept. For analysis of variance, Xs is the dummy variables from a categorical independent variable. X can represent interaction term or polynomial term to represent nonlinear relationship.

Let's create an example with only one predictor, X. Let $X \sim N(0, 1)$, $\beta = 0.5$, $\alpha = 2$, and $e \sim N(0, \sqrt{0.75})$. Then, this model can be used to generate data with sample size of 100 in R as follows:

```
> set.seed(123321)
> n <- 100
> x <- rnorm(n, 0, 1)
> e <- rnorm(n, 0, sqrt(0.75))
> y <- 2 + 0.5*x + e
```

As another example, regression with three predictors are created. Let X_1, X_2, and X_3 be multivariate normally distributed as follows:

$$\begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} \sim N\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0.5 & 0.2 \\ 0.5 & 1 & 0.3 \\ 0.2 & 0.3 & 1 \end{bmatrix} \right) \tag{2}$$

Let $\beta_1 = 0.2$, $\beta_2 = 0.3$, $\beta_3 = 0.4$, $\alpha = 1$, and $e \sim N(0, \sqrt{0.546})$. Then, this model can be used to generate data with sample size of 100 in R as follows:

```
> set.seed(123321)
> n <- 100
> MX <- rep(0, 3)
> SX <- matrix(1, 3, 3)
> SX[1, 2] <- SX[2, 1] <- 0.5
> SX[1, 3] <- SX[3, 1] <- 0.2
> SX[2, 3] <- SX[3, 2] <- 0.3
> library(MASS)
> X <- mvrnorm(n, MX, SX)
> e <- rnorm(n, 0, sqrt(0.546))
> y <- 1 + 0.2 * X[,1] + 0.3 * X[,2] + 0.4 * X[,3] + e
```

Multiple regression model above can be written in a matrix form.

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{e}, \tag{3}$$

where $\mathbf{y} = \begin{bmatrix} y_1 & y_2 & \dots & y_n \end{bmatrix}'$, $\boldsymbol{\beta} = \begin{bmatrix} \beta_0 & \beta_1 & \dots & \beta_p \end{bmatrix}'$, $\mathbf{e} = \begin{bmatrix} e_1 & e_2 & \dots & e_n \end{bmatrix}'$, $\mathbf{X} = \begin{bmatrix} \mathbf{1}_n & \mathbf{x}_1 & \dots & \mathbf{x}_p \end{bmatrix}$, $p =$ the number of predictors, $n =$ the number of observations, $\mathbf{1}_n$ is the vector of 1 with the length of n, and $\mathbf{x}_p = \begin{bmatrix} x_{1p} & x_{2p} & \dots & x_{np} \end{bmatrix}'$.

In the example of simple regression above, $\boldsymbol{\beta} = \begin{bmatrix} 2 & 0.5 \end{bmatrix}'$, $\mathbf{X} = \begin{bmatrix} \mathbf{1}_n & \mathbf{x} \end{bmatrix}'$, $\mathbf{x} \sim N(0, 1)$, and $\mathbf{e} \sim N(0, \sqrt{0.75})$. Data can be generated by matrix notation as follows:

```
> set.seed(123321)
> X <- matrix(1, n, 2)
> X[,2] <- rnorm(n, 0, 1)
> beta <- matrix(c(2, 0.5))
> e <- matrix(rnorm(n, 0, sqrt(0.75)))
> y <- X %*% beta + e
```

Because this equation has only one predictor, `X` is a matrix with n rows and 2 columns (the vector of one and the predictor). `beta` is a vector for the intercept and the slope respectively. It can be changed to the matrix format by using the `matrix` function. In the last line, `%*%` is the operator for matrix multiplication.

In the example of multiple regression with three predictors above, $\boldsymbol{\beta} = \begin{bmatrix} 1 & 0.2 & 0.3 & 0.4 \end{bmatrix}'$, $\mathbf{X} = \begin{bmatrix} \mathbf{1}_n & \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 \end{bmatrix}$, $\mathbf{e} \sim N(0, \sqrt{0.546})$, and $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$ are generated from Equation 2. Data can be generated by matrix notation as follows:

```
> set.seed(123321)
> n <- 100
> MX <- rep(0, 3)
> SX <- matrix(1, 3, 3)
> SX[1, 2] <- SX[2, 1] <- 0.5
> SX[1, 3] <- SX[3, 1] <- 0.2
> SX[2, 3] <- SX[3, 2] <- 0.3
> library(MASS)
> X <- cbind(1, mvrnorm(n, MX, SX))
> e <- matrix(rnorm(n, 0, sqrt(0.546)))
> beta <- matrix(c(1, 0.2, 0.3, 0.4))
> y <- X %*% beta + e
```

Next, let's generate data from a model with two-way interaction as follows:

$$Y = \alpha + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_1 X_2 + e \tag{4}$$

where X_1 and X_2 are multivariate normally distributed with the means of 0, the variances of 1, and the correlation of .2. $\beta_1 = 0.3$, $\beta_2 = 0$, $\beta_3 = 0.1$, $e \sim N(0, 1)$, $\alpha = 2$. Then, the data can be generated as follows:

```
> set.seed(123321)
> n <- 100
> MX <- rep(0, 2)
> SX <- matrix(1, 2, 2)
> SX[1, 2] <- SX[2, 1] <- 0.2
> library(MASS)
> X <- cbind(1, mvrnorm(n, MX, SX))
> X <- cbind(X, X[,2] * X[,3])
> e <- matrix(rnorm(n, 0, 1))
> beta <- matrix(c(2, 0.3, 0, 0.1))
> y <- X %*% beta + e
> dat <- data.frame(y = y, x1 = X[,2], x2 = X[,3])
```

Finally, let's generate data for independent t-test using multiple regression. Let Y_1 and Y_2 are the scores from each group. $Y_1 \sim N(50, 10)$ and $Y_2 \sim N(45, 10)$. The sample size for each group is 50. If we create a dummy variable that 1 represents Group 1 and 0 represents Group 2, $\alpha = 45$, $\beta_1 = 5$, and $e \sim N(0, 10)$.

```
> set.seed(123321)
> n <- 100
> X <- rep(c(0, 1), each = 50)
> X <- cbind(1, X)
> e <- matrix(rnorm(n, 0, 10))
> beta <- matrix(c(45, 5))
> y <- X %*% beta + e
```

Using the regression model, the variances within each group must be equal. To generate data with different variances, it is easier to generate data from each group separately.

```
> set.seed(123321)
> y1 <- rnorm(50, 50, 10)
> y2 <- rnorm(50, 45, 10)
```

You may be wondering why do we need to know how to generate data from dummy variables. For regression model, generating data from different groups is much easier. However, it is not convenient in more complex models, such as multilevel model in Section 1.4.

1.2 Logistic Regression

When the dependent variable is not continuous, multiple regression cannot be used. Logistic regression is designed to run a model with categorical outcomes. First, the logistic model for binary outcome is as follows:

$$\log\left(\frac{\Pr(Y=1)}{1-\Pr(Y=1)}\right) = \beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p \tag{5}$$

where Y is a binary outcome with two possible categories (0 or 1), $\Pr()$ is the probability function. This equation can be rewritten as

$$\Pr(Y=1) = \frac{\exp(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p)}{1+\exp(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p)} = \frac{1}{1+\exp(-(\beta_0 + \beta_1 x_1 + \cdots + \beta_p x_p))} \tag{6}$$

For example, let this logistic regression has three predictors, x_1, x_2, and x_3 which are normally distributed following Equation 2. $b_0 = 1$, $b_1 = 0.2$, $b_2 = 0.3$, and $b_3 = 0.4$. Then, data generation can be implemented in R as follows:

```
> set.seed(123321)
> n <- 100
> MX <- rep(0, 3)
> SX <- matrix(1, 3, 3)
> SX[1, 2] <- SX[2, 1] <- 0.5
> SX[1, 3] <- SX[3, 1] <- 0.2
> SX[2, 3] <- SX[3, 2] <- 0.3
> library(MASS)
> X <- cbind(1, mvrnorm(n, MX, SX))
> pred <- 1 + 0.2 * X[,1] + 0.3 * X[,2] + 0.4 * X[,3]
> predprob <- 1/(1 + exp(-pred))
> y <- rbinom(n, 1, predprob)
```

Logistic regression is also designed for multinomial dependent variables (i.e., more than 2 category). Let y has J category and the J-th category is the baseline. Then baseline-category logistic model is

$$\log\left(\frac{\Pr(Y=j)}{\Pr(Y=J)}\right) = \alpha_j + \beta_{1j}X_1 + \cdots + \beta_{pj}X_p \tag{7}$$

where $j = 1, \ldots, J-1$. Alternatively,

$$\frac{\Pr(Y=j)}{\Pr(Y=J)} = \exp\left(\alpha_j + \beta_{1j}X_1 + \cdots + \beta_{pj}X_p\right) \tag{8}$$

From here, the proportion of each category can be calculated by

$$\Pr(Y=j) = \frac{\Pr(Y=j)/\Pr(Y=J)}{\Pr(Y=1)/\Pr(Y=J) + \Pr(Y=2)/\Pr(Y=J) + \ldots + \Pr(Y=J)/\Pr(Y=J)} \tag{9}$$

Note that $\Pr(Y=J)/\Pr(Y=J) = 1$.

As an example, let Y has 3 categories and the third category is the baseline. Then, there are two equations explaining the log odds of Category 1 vs Category 3 and Category 2 vs Category 3. Let this model has only one predictor, x, where $x \sim N(0,1)$. $\alpha_1 = 0.5$, $\alpha_2 = 1$, $\beta_1 = 0.5$, and $\beta_2 = 0.3$. Then, data with 100 observations can be simulated by R as follows:

```
> set.seed(123321)
> n <- 100
> x <- rnorm(n, 0, 1)
> yhat1 <- 0.5 + 0.5*x
> yhat2 <- 1 + 0.3*x
> p1overp3 <- exp(yhat1)
> p2overp3 <- exp(yhat2)
> p1 <- p1overp3 / (p1overp3 + p2overp3 + 1)
> p2 <- p2overp3 / (p1overp3 + p2overp3 + 1)
> p3 <- 1 / (p1overp3 + p2overp3 + 1)
> probs <- cbind(p1, p2, p3)
> drawmult <- function(p) which(as.logical(rmultinom(1, size = 1, prob = p)))
> y <- apply(probs, 1, drawmult)
```

Logistic regression can also be used to deal with an ordinal dependent variable. There are many models for ordinal responses. The model introduced here is cumulative logit model. Let y has J ordered categories. The cumulative logistic model is

$$\log\left(\frac{\Pr(Y \le j)}{1 - \Pr(Y \le j)}\right) = \alpha_j + \beta_1 X_1 + \cdots + \beta_p X_p \tag{10}$$

where $j = 1, \ldots, J-1$ and $\alpha_1 < \alpha_2 < \ldots < \alpha_{J-1}$.

Alternatively,

$$\Pr(Y \le j) = \frac{1}{1 + \exp\left(-(\alpha_j + \beta_1 X_1 + \cdots + \beta_p X_p)\right)} \tag{11}$$

From here, the proportions of each category can be calculated by

$$
\begin{aligned}
\Pr(Y = 1) &= \Pr(Y \leq 1) \\
\Pr(Y = 2) &= \Pr(Y \leq 2) - \Pr(Y \leq 1) \\
&\vdots \\
\Pr(Y = J) &= 1 - \Pr(Y \leq J - 1)
\end{aligned}
\tag{12}
$$

As an example, let Y has 5 categories. Then, there are four equations explaining the log odds of $\Pr(Y \leq 1)$, $\Pr(Y \leq 2)$, $\Pr(Y \leq 3)$, and $\Pr(Y \leq 4)$. Let this model has only one predictor, x, which $x \sim N(0, 1)$. $\alpha_1 = -2$, $\alpha_2 = -1$, $\alpha_3 = 0.5$, $\alpha_4 = 1.5$, and $\beta = 0.5$. Then, data with 100 observations can be simulated by R as follows:

```
> set.seed(123321)
> n <- 100
> x <- rnorm(n, 0, 1)
> yhat1 <- -2 + 0.5*x
> yhat2 <- -1 + 0.5*x
> yhat3 <- 0.5 + 0.5*x
> yhat4 <- 1.5 + 0.5*x
> pleq1 <- 1 / (1 + exp(-yhat1))
> pleq2 <- 1 / (1 + exp(-yhat2))
> pleq3 <- 1 / (1 + exp(-yhat3))
> pleq4 <- 1 / (1 + exp(-yhat4))
> p1 <- pleq1
> p2 <- pleq2 - pleq1
> p3 <- pleq3 - pleq2
> p4 <- pleq4 - pleq3
> p5 <- 1 - pleq4
> probs <- cbind(p1, p2, p3, p4, p5)
> drawmult <- function(p) which(as.logical(rmultinom(1, size = 1, prob = p)))
> y <- apply(probs, 1, drawmult)
```

1.3 Probit Model

Another way to create categorical variables depending on other variables is to use probit model. Probit model is usually used in structural equation modeling for categorical data. First, the probit model for dichotomous outcome is as follows:

$$
\begin{aligned}
\mu &= \beta_1 X_1 + \cdots + \beta_p X_p \\
\Pr(Y = 0) &= \Phi_{N(\mu,1)}(t) \\
\Pr(Y = 1) &= 1 - \Pr(Y = 0)
\end{aligned}
$$

where t is the threshold that separates scores coded as 0 and 1 and $\Phi_{N(a,b)}(x)$ is the cumulative distribution function (CDF) of the normal distribution with the mean of a and standard

deviation of b. That is, this function will find the proportion of area below the threshold (t) in the normal distribution with the mean of a and standard deviation of b.

For example, let this probit model has three predictors, x_1, x_2, and x_3 which are normally distributed following Equation 2. $t = 1$, $\beta_1 = 0.2$, $\beta_2 = 0.3$, and $\beta_3 = 0.4$. Then, data generation can be implemented in R as follows:

```
> set.seed(123321)
> n <- 100
> MX <- rep(0, 3)
> SX <- matrix(1, 3, 3)
> SX[1, 2] <- SX[2, 1] <- 0.5
> SX[1, 3] <- SX[3, 1] <- 0.2
> SX[2, 3] <- SX[3, 2] <- 0.3
> library(MASS)
> X <- cbind(1, mvrnorm(n, MX, SX))
> pred <- 0.2 * X[,1] + 0.3 * X[,2] + 0.4 * X[,3]
> p1 <- 1 - pnorm(1, mean = pred, sd = 1)
> y <- rbinom(n, 1, p1)
```

Probit model is also applicable with ordinal variables. Let y has J ordered category. The probit model for ordinal data is

$$\begin{aligned} \mu &= \beta_1 X_1 + \cdots + \beta_p X_p \\ \Pr\left(Y \leq j\right) &= \Phi_{N(\mu,1)}\left(t_j\right) \end{aligned}$$

where $j = 1, \ldots, J-1$. See Equation 12 for how to get the proportion of each category.

As an example, let Y has 5 categories. Then, there are four equations to find $\Pr(Y \leq 1)$, $\Pr(Y \leq 2)$, $\Pr(Y \leq 3)$, and $\Pr(Y \leq 4)$. Let this model has only one predictor, x, where $x \sim N(0, 1)$. $t_1 = -2$, $t_2 = -1$, $t_3 = 0.5$, $t_4 = 1.5$, and $\beta = 0.5$. Then, data with 100 observations can be simulated by R as follows:

```
> set.seed(123321)
> n <- 100
> x <- rnorm(n, 0, 1)
> pred <- 0.5*x
> pleq1 <- pnorm(-2, mean = pred, sd = 1)
> pleq2 <- pnorm(-1, mean = pred, sd = 1)
> pleq3 <- pnorm(0.5, mean = pred, sd = 1)
> pleq4 <- pnorm(1.5, mean = pred, sd = 1)
> p1 <- pleq1
> p2 <- pleq2 - pleq1
> p3 <- pleq3 - pleq2
> p4 <- pleq4 - pleq3
> p5 <- 1 - pleq4
> probs <- cbind(p1, p2, p3, p4, p5)
```

```
> drawmult <- function(p) which(as.logical(rmultinom(1, size = 1, prob = p)))
> y <- apply(probs, 1, drawmult)
```

1.4 Multilevel Model

Sometimes, data are in nested structure, such as students in classrooms. Multilevel model investigates the influences of the variables at both levels. In this class, I will refer the upper level as classrooms and the lower level as students. The most basic multilevel models is the random-effect analysis of variance model as follows:

$$Y_{ij} = \beta_{0j} + r_{ij}$$
$$\beta_{0j} = \gamma_{00} + u_{0j}$$

where i is the index of students, j is the index of classrooms, Y_{ij} is the dependent variable score of student i in classroom j, β_{0j} is the average score of dependent variable in classroom j, r_{ij} is the student-level residual, γ_{00} is the mean of the dependent variable across classrooms, and u_{0j} is the classroom-level residual. Note that this model is also called as null model because there is no predictor in this model.

This example will generate data for 10 classrooms with 5 students each. $r_{ij} \sim N(0, 1)$, $u_{0j} \sim N(0, 0.5)$, and $\gamma_{00} = 3$. Data generation can be done in R as follows:

```
> n <- 5
> k <- 10
> gamma00 <- 3
> u <- rnorm(k, 0, 0.5)
> beta0 <- gamma00 + u
> beta0 <- rep(beta0, each = n)
> r <- rnorm(n * k, 0, 1)
> y <- beta0 + r
> g <- rep(1:k, each = n)
> dat <- data.frame(y, g)
```

Let's create another example with unequal group size. All parameters are the same as the previous example. However, the group size of 10 classrooms are 4, 5, 6, 4, 5, 7, 4, 5, 6, and 7. Data generation can be done in R as follows:

```
> n <- c(4, 5, 6, 4, 5, 7, 4, 5, 6, 7)
> k <- 10
> gamma00 <- 3
> u <- rnorm(k, 0, 0.5)
> beta0 <- gamma00 + u
> beta0 <- mapply(rep, beta0, n)
> beta0 <- unlist(beta0)
> r <- rnorm(sum(n), 0, 1)
> y <- beta0 + r
```

```
> g <- mapply(rep, 1:k, n)
> g <- unlist(g)
> dat <- data.frame(y, g)
```

The next model adds classroom-level predictors to predict the intercepts as follows:

$$
\begin{aligned}
Y_{ij} &= \beta_{0j} + r_{ij} \\
\beta_{0j} &= \gamma_{00} + \gamma_{01} W_{1j} + \ldots + \gamma_{0q} W_{qj} + u_{0j}
\end{aligned}
$$

where q is the number of classroom-level predictors, γ_{0q} is the regression coefficient of the classroom-level predictor q, and W_{qj} is the value of the classroom-level predictor q from classroom j.

This example will generate data for 10 classrooms with 5 students each. One predictor is used in this model. It is a dummy variables that the first half of classrooms is 1 and the second half is 0. $r_{ij} \sim N(0, 1)$, $u_{0j} \sim N(0, 0.5)$, $\gamma_{00} = 3$, and $\gamma_{01} = 1.2$. Data generation can be done in R as follows:

```
> n <- 5
> k <- 10
> u <- rnorm(k, 0, 0.5)
> w <- rep(c(1, 0), each = k/2)
> beta0 <- 3 + 1.2 * w + u
> beta0 <- rep(beta0, each = n)
> r <- rnorm(n * k, 0, 1)
> y <- beta0 + r
> g <- rep(1:k, each = n)
> w <- rep(w, each = n)
> dat <- data.frame(y, g, w)
```

The next model will add student-level predictors from the random-effect analysis of variance model (null model). The regression coefficients of the student-level predictors are not constant across classrooms.

$$
\begin{aligned}
Y_{ij} &= \beta_{0j} + \beta_{1j} X_{1ij} + \ldots + \beta_{pj} X_{pij} + r_{ij} \\
\beta_{0j} &= \gamma_{00} + u_{0j} \\
\beta_{1j} &= \gamma_{10} + u_{1j} \\
&\vdots \\
\beta_{pj} &= \gamma_{p0} + u_{pj}
\end{aligned}
$$

where p is the number of student-level predictors, X_{pij} is the value of the student-level predictor p from student i in classroom j, β_{pj} is the regression coefficient of the student-level predictor p in classroom j, γ_{p0} is the average value of the regression coefficient of

the student-level predictor p across classrooms, and u_{pj} is the student-level residual of the regression coefficient of student-level predictor p.

This example will generate data for 10 classrooms with 5 students each again. One student-level predictor is used in this model where $X \sim N(\mu_{Xj}, 1)$ and $\mu_{Xj} \sim N(0, 0.5)$. This data generation allows X to have different means across classrooms. $\gamma_{00} = 3$, $\gamma_{10} = 0.2$, $r_{ij} \sim N(0, 1)$, and $\begin{bmatrix} u_{0j} \\ u_{1j} \end{bmatrix} \sim N\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.5 & 0.01 \\ 0.01 & 0.02 \end{bmatrix}\right)$. Data generation can be done in R as follows:

```
> n <- 5
> k <- 10
> library(MASS)
> TAU <- matrix(c(0.5, 0.01, 0.01, 0.02), 2, 2)
> U <- mvrnorm(k, rep(0, 2), TAU)
> beta0 <- 3 + U[,1]
> beta1 <- 0.2 + U[,2]
> beta0 <- rep(beta0, each = n)
> beta1 <- rep(beta1, each = n)
> r <- rnorm(n * k, 0, 1)
> mux <- rnorm(k, 0, 0.5)
> mux <- rep(mux, each = n)
> x <- rnorm(n * k, mux, 1)
> y <- beta0 + beta1 * x + r
> g <- rep(1:k, each = n)
> dat <- data.frame(y, g, x)
```

Finally, the full multilevel model has both student-level and classroom-level predictors. The model is as follows:

$$
\begin{aligned}
Y_{ij} &= \beta_{0j} + \beta_{1j}X_{1ij} + \ldots + \beta_{pj}X_{pij} + r_{ij} \\
\beta_{0j} &= \gamma_{00} + \gamma_{01}W_{1j} + \ldots + \gamma_{0q}W_{qj} + u_{0j} \\
\beta_{1j} &= \gamma_{10} + \gamma_{11}W_{1j} + \ldots + \gamma_{1q}W_{qj} + u_{1j} \\
&\vdots \\
\beta_{pj} &= \gamma_{p0} + \gamma_{p1}W_{1j} + \ldots + \gamma_{pq}W_{qj} + u_{pj}
\end{aligned}
$$

where γ_{pq} is the effect of the classroom-level predictor q on the slope of the student-level predictor p. This effect is also referred to as cross-level interaction.

This example will generate data for 10 classrooms with 5 students each again. There are one student-level predictor X and one classroom-level predictor W. One student-level predictor is used in this model where $X \sim N(\mu_{Xj}, 1)$ and $\mu_{Xj} \sim N(0, 0.5)$. W is a dummy variables that the first half of classrooms is 1 and the second half is 0. $\gamma_{00} = 2.6$, $\gamma_{10} = 0.18$, $\gamma_{01} = 0.8$, $\gamma_{11} = 0.04$, $r_{ij} \sim N(0, 1)$, and $\begin{bmatrix} u_{0j} \\ u_{1j} \end{bmatrix} \sim N\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0.5 & 0.01 \\ 0.01 & 0.02 \end{bmatrix}\right)$. Data generation can be done in R as follows:

```
> n <- 5
> k <- 10
> w <- rep(c(1, 0), each = k/2)
> library(MASS)
> TAU <- matrix(c(0.5, 0.01, 0.01, 0.02), 2, 2)
> U <- mvrnorm(k, rep(0, 2), TAU)
> beta0 <- 2.6 + 0.8*w + U[,1]
> beta1 <- 0.18 + 0.04*w + U[,2]
> beta0 <- rep(beta0, each = n)
> beta1 <- rep(beta1, each = n)
> r <- rnorm(n * k, 0, 1)
> mux <- rnorm(k, 0, 0.5)
> mux <- rep(mux, each = n)
> x <- rnorm(n * k, mux, 1)
> y <- beta0 + beta1 * x + r
> g <- rep(1:k, each = n)
> w <- rep(w, each = n)
> dat <- data.frame(y, g, x, w)
```

Note that longitudinal data can be created by multilevel framework as well. The lower level represents time and the upper level is individual. Time will be used as a lower-level predictor. Alternatively, longitudinal data can be generated using structural equation modelling (SEM). SEM provides many models for longitudinal data, such as autoregressive models or latent growth curve. If SEM is used, the `simulateData` function in the `lavaan` package is very useful. Users can simply specify the values of parameters of the `lavaan` model and the number of observations to generate data by this function.

2 Impose Missing Data

In this section, we will cover how to impose missing data. In most missing data literature, missing data process is seperated into three types: missing completely at random (MCAR), missing at random (MAR), and not missing at random (NMAR).

2.1 Missing Completely at Random

Missing completely at random means that missing data process does not depend on the values of missing variable and the values of other observed and unobserved variables. For example, X_1, X_2, X_3, and Z are observed variables. All variables have variance of 1. The correlations among Xs are .5 and the correlations between Xs and Z are .4. The means of all variables are 0. The probability of missing values of X_1 is .20.

```
> set.seed(123321)
> library(MASS)
> n <- 100
> S <- matrix(0.5, 4, 4)
```

```
> S[4,] <- 0.4
> S[,4] <- 0.4
> diag(S) <- 1
> dat <- mvrnorm(n, rep(0, 4), S)
> colnames(dat) <- c("x1", "x2", "x3", "z")
> mcardat <- dat
> mcardat[as.logical(rbinom(n, 1, 0.2)), 1] <- NA
> summary(mcardat)
```

```
       x1                 x2                 x3                 z
 Min.   :-2.3806   Min.   :-1.96055   Min.   :-2.675013   Min.   :-2.325727
 1st Qu.:-0.7567   1st Qu.:-0.84344   1st Qu.:-0.642963   1st Qu.:-0.757997
 Median :-0.3141   Median :-0.15768   Median :-0.052697   Median :-0.149174
 Mean   :-0.2655   Mean   :-0.07173   Mean   : 0.009089   Mean   : 0.009678
 3rd Qu.: 0.4548   3rd Qu.: 0.54318   3rd Qu.: 0.640105   3rd Qu.: 0.603701
 Max.   : 1.7836   Max.   : 2.88555   Max.   : 3.230012   Max.   : 2.772292
 NA's   :23
```

2.2 Missing at Random

Missing at random means that missing data process of one variable depends on the values of other variables. The other variables must be observed and accounted for in your data analysis so that the bias from missing data is minimized. From the previous data set, the missing value of X_1 depends on the value of Z. Because missing values can be indicated by binary variable (yes vs no), logistic regression or probit model can be used to provide the relationship between the missing values of X_1 and Z. For example, logistic regression is used here.

```
> mardat <- dat
> z <- dat[,4]
> pred <- -1.386 + 0.288 * z
> predprob <- 1/(1 + exp(-pred))
> mardat[as.logical(rbinom(n, 1, predprob)), 1] <- NA
> summary(mardat)
```

```
       x1                 x2                 x3                 z
 Min.   :-2.3806   Min.   :-1.96055   Min.   :-2.675013   Min.   :-2.325727
 1st Qu.:-0.8757   1st Qu.:-0.84344   1st Qu.:-0.642963   1st Qu.:-0.757997
 Median :-0.2962   Median :-0.15768   Median :-0.052697   Median :-0.149174
 Mean   :-0.2155   Mean   :-0.07173   Mean   : 0.009089   Mean   : 0.009678
 3rd Qu.: 0.4422   3rd Qu.: 0.54318   3rd Qu.: 0.640105   3rd Qu.: 0.603701
 Max.   : 2.7691   Max.   : 2.88555   Max.   : 3.230012   Max.   : 2.772292
 NA's   :20
```

To get the expected probability of missing value, we need to do some algebras to solve for the appropriate values of β_0 and β_1 in logistic regression. For example, if we want to have

20% chance of missing when $z = 0$ and 25% chance of missing when $z = 1$, we can substitute these values in Equation 5 to get two equations solving for two unknowns.

$$\log\left(\frac{0.20}{1-0.20}\right) = \beta_0 + \beta_1(0)$$
$$\log\left(\frac{0.25}{1-0.25}\right) = \beta_0 + \beta_1(1)$$

Then, $\beta_0 = -1.386$ and $\beta_1 = 0.288$.

2.3 Missing Not at Random

When missing values are not at random, there are two possiblities. First, the values that determine the process of missing values are not observed. For example, the missingness of X_1 depends on Z but Z is not observed or appropriately accounted for in data analysis. Second, the missing values depend on the values of the variable itself. For example, the missingness of X_1 depends on the values of X_1 as the following example:

```
> nmardat <- dat
> pred <- -1.386 + 0.288 * nmardat[,1]
> predprob <- 1/(1 + exp(-pred))
> nmardat[as.logical(rbinom(n, 1, predprob)), 1] <- NA
> summary(nmardat)
```

```
       x1                 x2                 x3                  z
 Min.   :-2.3806   Min.   :-1.96055   Min.   :-2.675013   Min.   :-2.325727
 1st Qu.:-0.9402   1st Qu.:-0.84344   1st Qu.:-0.642963   1st Qu.:-0.757997
 Median :-0.3167   Median :-0.15768   Median :-0.052697   Median :-0.149174
 Mean   :-0.2559   Mean   :-0.07173   Mean   : 0.009089   Mean   : 0.009678
 3rd Qu.: 0.4059   3rd Qu.: 0.54318   3rd Qu.: 0.640105   3rd Qu.: 0.603701
 Max.   : 2.7691   Max.   : 2.88555   Max.   : 3.230012   Max.   : 2.772292
 NA's   :18
```

2.4 Attrition

When participants drop out of a study, researchers cannot obtain any data in the future. For example, if a participant drops out at Time 2, researchers will not have data from Time 3 or later from this participant. In this section, the example data set is simulated from simplex model with the first-order autoregression coefficient of 0.5 for four time points. Three missing data patterns are covered here. The first example of missing pattern is that the process of missing data is MCAR.

```
> t1 <- rnorm(n, 0, 1)
> t2 <- 0.5*t1 + rnorm(n, 0, sqrt(0.75))
> t3 <- 0.5*t1 + rnorm(n, 0, sqrt(0.75))
```

```
> t4 <- 0.5*t1 + rnorm(n, 0, sqrt(0.75))
> m2 <- as.logical(rbinom(n, 1, 0.05))
> m3 <- as.logical(rbinom(n, 1, 0.05)) | m2
> m4 <- as.logical(rbinom(n, 1, 0.05)) | m3
> t2[m2] <- NA
> t3[m3] <- NA
> t4[m4] <- NA
> dat <- data.frame(t1, t2, t3, t4)
> summary(dat)
```

```
       t1                  t2                  t3                  t4
 Min.   :-2.141952   Min.   :-2.32118   Min.   :-1.5969   Min.   :-2.4572
 1st Qu.:-0.654138   1st Qu.:-0.68583   1st Qu.:-0.4889   1st Qu.:-0.4323
 Median : 0.008073   Median :-0.10221   Median : 0.1400   Median : 0.3249
 Mean   : 0.029303   Mean   :-0.05509   Mean   : 0.1990   Mean   : 0.2876
 3rd Qu.: 0.668795   3rd Qu.: 0.52819   3rd Qu.: 0.8874   3rd Qu.: 0.9695
 Max.   : 2.753779   Max.   : 3.76914   Max.   : 2.7900   Max.   : 2.2302
                     NA's   :7          NA's   :10        NA's   :20
```

Notice that | is used in creating m3 and m4. For m3, if any observations are missing at Time 2, the observations are missing at Time 3 automatically because | is used to determine the missing at Time 3. For the nonmissing observations at Time 2, the probability of missing at Time 3 is 5%.

The second example of missing pattern is that the rate of dropout depends on the characteristics of participants (e.g., age), which is also called time-invariant covariates. Here, X is the participant characteristic where $X \sim N(0, 1)$. The rates of dropout are 5% and 10% if $X = 0$ and 2, respectively.

```
> t1 <- rnorm(n, 0, 1)
> t2 <- 0.5*t1 + rnorm(n, 0, sqrt(0.75))
> t3 <- 0.5*t1 + rnorm(n, 0, sqrt(0.75))
> t4 <- 0.5*t1 + rnorm(n, 0, sqrt(0.75))
> x  <- rnorm(n, 0, 1)
> pred <- -2.944 + 0.374 * x
> predprob <- 1/(1 + exp(-pred))
> m2 <- as.logical(rbinom(n, 1, predprob))
> m3 <- as.logical(rbinom(n, 1, predprob)) | m2
> m4 <- as.logical(rbinom(n, 1, predprob)) | m3
> t2[m2] <- NA
> t3[m3] <- NA
> t4[m4] <- NA
> dat <- data.frame(t1, t2, t3, t4, x)
> summary(dat)
```

```
       t1                  t2                  t3                  t4
 Min.   :-2.054764   Min.   :-1.97324   Min.   :-3.753302   Min.   :-1.73984
```

```
 1st Qu.:-0.705874   1st Qu.:-0.65885   1st Qu.:-0.788036   1st Qu.:-0.70691
 Median : 0.007116   Median :-0.01183   Median :-0.008816   Median :-0.06203
 Mean   :-0.039591   Mean   :-0.04737   Mean   :-0.027561   Mean   :-0.05612
 3rd Qu.: 0.630845   3rd Qu.: 0.50138   3rd Qu.: 0.813481   3rd Qu.: 0.47030
 Max.   : 1.731586   Max.   : 2.25608   Max.   : 2.364722   Max.   : 1.81946
                     NA's   :6          NA's   :11          NA's   :17
       x
 Min.   :-2.37145
 1st Qu.:-0.60959
 Median :-0.06756
 Mean   :-0.02197
 3rd Qu.: 0.53136
 Max.   : 2.12949
```

The third example of missing pattern is that the rate of dropout depends on a variable at each time point, which is also called time-varying covariates. Let Z be the time-varying covariate created from autoregressive model with the first-order autoregression coefficient of .3. The rates of dropout are 5% and 10% if $Z = 0$ and 2, respectively.

```
> t1 <- rnorm(n, 0, 1)
> t2 <- 0.5*t1 + rnorm(n, 0, sqrt(0.75))
> t3 <- 0.5*t1 + rnorm(n, 0, sqrt(0.75))
> t4 <- 0.5*t1 + rnorm(n, 0, sqrt(0.75))
> z1 <- rnorm(n, 0, 1)
> z2 <- 0.3*t1 + rnorm(n, 0, sqrt(0.91))
> z3 <- 0.3*t1 + rnorm(n, 0, sqrt(0.91))
> z4 <- 0.3*t1 + rnorm(n, 0, sqrt(0.91))
> pred1 <- -2.944 + 0.374 * z1
> pred2 <- -2.944 + 0.374 * z2
> pred3 <- -2.944 + 0.374 * z3
> pred4 <- -2.944 + 0.374 * z4
> predprob1 <- 1/(1 + exp(-pred1))
> predprob2 <- 1/(1 + exp(-pred2))
> predprob3 <- 1/(1 + exp(-pred3))
> predprob4 <- 1/(1 + exp(-pred4))
> m1 <- as.logical(rbinom(n, 1, predprob1))
> m2 <- as.logical(rbinom(n, 1, predprob2)) | m1
> m3 <- as.logical(rbinom(n, 1, predprob3)) | m2
> m4 <- as.logical(rbinom(n, 1, predprob4)) | m3
> t1[m1] <- NA
> t2[m2] <- NA
> t3[m3] <- NA
> t4[m4] <- NA
> dat <- data.frame(t1, t2, t3, t4, z1, z2, z3, z4)
> summary(dat)
```

```
      t1                  t2                  t3                  t4
 Min.   :-2.83484   Min.   :-2.04000   Min.   :-2.18892   Min.   :-2.7412
 1st Qu.:-0.77175   1st Qu.:-0.56208   1st Qu.:-0.67909   1st Qu.:-0.6945
 Median :-0.05479   Median : 0.05356   Median :-0.13054   Median :-0.1238
 Mean   :-0.08419   Mean   : 0.11216   Mean   :-0.02155   Mean   :-0.1734
 3rd Qu.: 0.52670   3rd Qu.: 1.00189   3rd Qu.: 0.44668   3rd Qu.: 0.4205
 Max.   : 2.77290   Max.   : 2.31050   Max.   : 2.79372   Max.   : 2.1584
 NA's   :7          NA's   :15         NA's   :16         NA's   :24
      z1                  z2                  z3                  z4
 Min.   :-2.40888   Min.   :-3.551314   Min.   :-2.63571   Min.   :-1.868155
 1st Qu.:-0.71708   1st Qu.:-0.661954   1st Qu.:-0.75688   1st Qu.:-0.682191
 Median : 0.07499   Median : 0.017187   Median : 0.00338   Median : 0.009697
 Mean   :-0.03534   Mean   :-0.006197   Mean   :-0.02609   Mean   : 0.021509
 3rd Qu.: 0.64437   3rd Qu.: 0.754479   3rd Qu.: 0.70314   3rd Qu.: 0.667312
 Max.   : 2.34019   Max.   : 2.619284   Max.   : 2.17152   Max.   : 3.048968
```

3 Missing Data Handling

This section will provide how to handle missing data using different methods. This section will include both good and bad methods for handling missing data. To begin with, let's create a data set used in this section. This data set has 7 variables: 5 target variables ($X_1 - X_5$) and 2 auxiliary variables ($Z_1 - Z_2$). X_1 is a nominal variable with four categories. X_2 is an ordinal variable with four categories. X_3 is a continuous variable drawn from normal distribution with the mean of 0 and the standard deviaton of 1. X_4 is a continuous variable drawn from chi-square distribution with $df = 4$. X_5 is a binary variable with $\Pr(X_5 = 1) = .30$. Z_1 and Z_2 are normally distributed with the means of 0 and the variances of 1. All variables are independent because they are created independently.

The missing data of X_1 is MCAR. The missing data of X_2 is MAR depending on Z_1. The missing data of X_3 is MAR depending on Z_2. The missing data of X_4 is NMAR depending on itself. The missing data of X_5, Z_1, and Z_2 are MCAR.

```
> n <- 100
> x1 <- apply(rmultinom(n, size = 1, prob = c(0.2, 0.3, 0.4, 0.1)) == 1, 2,
+         which) # Nominal
> x2 <- apply(rmultinom(n, size = 1, prob = c(0.2, 0.3, 0.4, 0.1)) == 1, 2,
+         which) # Ordinal
> x3 <- rnorm(n, 0, 1)
> x4 <- rchisq(n, 4)
> x5 <- rbinom(n, 1, 0.3)
> z1 <- rnorm(n, 0, 1)
> z2 <- rnorm(n, 0, 1)
> predz1 <- -1 + 0.2 * z1
> predprobz1 <- 1/(1 + exp(-predz1))
> predz2 <- -2 + 0.3 * z2
```

```
> predprobz2 <- 1/(1 + exp(-predz2))
> predx4 <- -1 + 0.2 * x4
> predprobx4 <- 1/(1 + exp(-predx4))
> mx1 <- as.logical(rbinom(n, 1, 0.1))
> mx2 <- as.logical(rbinom(n, 1, predprobz1))
> mx3 <- as.logical(rbinom(n, 1, predprobz2))
> mx4 <- as.logical(rbinom(n, 1, predprobx4))
> mx5 <- as.logical(rbinom(n, 1, 0.05))
> mz1 <- as.logical(rbinom(n, 1, 0.05))
> mz2 <- as.logical(rbinom(n, 1, 0.20))
> x1[mx1] <- NA; x2[mx2] <- NA; x3[mx3] <- NA; x4[mx4] <- NA; x5[mx5] <- NA
> z1[mz1] <- NA; z2[mz2] <- NA
> dat <- data.frame(x1, x2, x3, x4, x5, z1, z2)
> summary(dat)

       x1              x2              x3                x4
 Min.   :1.000   Min.   :1.000   Min.   :-2.3956   Min.   : 0.2761
 1st Qu.:2.000   1st Qu.:2.000   1st Qu.:-0.5792   1st Qu.: 1.7280
 Median :2.000   Median :3.000   Median :-0.0608   Median : 3.2828
 Mean   :2.374   Mean   :2.429   Mean   :-0.0361   Mean   : 3.4398
 3rd Qu.:3.000   3rd Qu.:3.000   3rd Qu.: 0.5248   3rd Qu.: 4.6910
 Max.   :4.000   Max.   :4.000   Max.   : 2.7796   Max.   :12.0802
 NA's   :9       NA's   :30      NA's   :13        NA's   :43
       x5               z1                 z2
 Min.   :0.0000   Min.   :-2.300909   Min.   :-2.65084
 1st Qu.:0.0000   1st Qu.:-0.717535   1st Qu.:-0.68678
 Median :0.0000   Median : 0.007946   Median :-0.02568
 Mean   :0.3118   Mean   :-0.019845   Mean   :-0.03029
 3rd Qu.:1.0000   3rd Qu.: 0.661981   3rd Qu.: 0.79714
 Max.   :1.0000   Max.   : 2.699295   Max.   : 2.34773
 NA's   :7        NA's   :6           NA's   :21

> library(mice)
> md.pattern(dat)

   z1 x5 x1 x3 z2 x2 x4
19  1  1  1  1  1  1  1   0
 3  1  1  0  1  1  1  1   1
15  1  1  1  1  1  0  1   1
 3  1  1  1  0  1  1  1   1
15  1  1  1  1  1  1  0   1
 2  1  0  1  1  1  1  1   1
10  1  1  1  1  0  1  1   1
 3  1  1  0  1  1  1  0   2
 5  1  1  1  1  1  0  0   2
```

```
 2  1  1  1  0  1  1  0   2
 2  1  0  1  1  1  1  0   2
 2  0  1  1  1  1  0  1   2
 3  0  1  1  1  1  1  0   2
 2  1  1  1  1  0  0  1   2
 2  1  1  1  1  0  1  0   2
 1  1  1  0  1  1  0  0   3
 1  1  1  0  0  1  1  0   3
 3  1  1  1  0  1  0  0   3
 1  1  1  0  1  0  1  0   3
 1  1  1  1  1  0  0  0   3
 2  1  1  1  0  0  1  0   3
 1  1  0  1  1  0  0  0   4
 1  1  0  1  0  0  1  0   4
 1  0  0  1  0  0  1  1   4
    6  7  9 13 21 30 43 129
```

The `md.pattern` function can be used to find the missing data patterns in a target data set. In the following subsections, how to deal with missing data with different methods is introduced.

3.1 Listwise Deletion

Listwise deletion is to analyze data for cases without any missing observations. This method is bad because much information is lost. Deleted cases have partial information that is totally ignored by this method. This method is the default method for many functions in R, such as `lm`, `glm`, or `lavaan`.

```
> rowna <- apply(is.na(dat), 1, any)
> datlistwise <- dat[!rowna, ]
> dim(datlistwise)

[1] 19  7
```

3.2 Pairwise Deletion

This method is usually mentioned in correlation. In finding pairwise correlation for many variables, all cases that have the information of both variables are used. Thus, the number of cases calculating different pairs of correlations are different. The problem of this method is that the correlations could be inconsistent. The correlation matrix can be nonpositive definite.

```
> cor(dat, use = "pairwise.complete.obs")

            x1          x2          x3          x4          x5           z1
x1  1.00000000  0.16959634 -0.04221941  0.09542418 -0.01454993 -0.067912025
```

```
x2  0.16959634  1.00000000 -0.23413333  0.10197789  0.21560518  0.032738432
x3 -0.04221941 -0.23413333  1.00000000  0.01870129 -0.12485457  0.094136444
x4  0.09542418  0.10197789  0.01870129  1.00000000 -0.12303112 -0.024843927
x5 -0.01454993  0.21560518 -0.12485457 -0.12303112  1.00000000 -0.106137219
z1 -0.06791202  0.03273843  0.09413644 -0.02484393 -0.10613722  1.000000000
z2  0.04063691 -0.06582822 -0.06833536 -0.06349951 -0.09637512  0.004217929
             z2
x1  0.040636913
x2 -0.065828221
x3 -0.068335359
x4 -0.063499513
x5 -0.096375124
z1  0.004217929
z2  1.000000000
```

3.3 Mean Substitution

This method is simply to substitute missing observations by the mean of the variable from all available cases. Of course, this method is not good because it underestimates the variance of the variable. In this example, all variables except X_1 are imputed by their means. Imputing X_1 does not make sense because it is unordered categorical variable. Imputing X_2 actually does not make sense because it is ordered categorical variables. However, if the number of categories is high, some analysts assume ordered variables to be continuous.

```
> datmeansub <- dat
> targetvar <- 2:7
> for(i in targetvar) {
+   datmeansub[is.na(datmeansub[, i]), i] <- mean(datmeansub[, i],  na.rm = TRUE)
+ }
```

3.4 Group-mean Substitution

This method is to substitute missing observations by its group mean of all available cases. Group is defined by analysts in advance. This method is similar to regression analysis with grouping variable as a predictor. This method is not good because it overestimates the relationship between the grouping variable and the imputed variable. Again, all variables except X_1 are imputed by their group means defined by X_1.

```
> datgroupsub <- dat
> groupvar <- 1
> catgroupvar <- setdiff(unique(dat[,groupvar]), NA)
> targetvar <- 2:7
> for(i in targetvar) {
+   for(j in 1:length(catgroupvar)) {
+     subrow <- is.na(datgroupsub[, i]) &
```

```
+                 sapply(datgroupsub[,groupvar] == catgroupvar[j], isTRUE)
+     calcrow <- datgroupsub[,groupvar] == catgroupvar[j]
+     datgroupsub[subrow, i] <- mean(datgroupsub[calcrow, i],  na.rm = TRUE)
+   }
+ }
```

3.5 Row-mean Substitution

This method is to impute missing values by the row mean of all other variables in the same scale. This method is not good because it is similar to regression that weighs all variables equally, which is only possible when all items in a scale are parallel. Further, it treats the imputed value as if it is observed–applicable for all mean substitution method, although they are not actually observed. The sample size is overestimated so the standard errors are lower than they should be.

The new data set is used in this example. This example has five variables: Y_1, Y_2, Y_3, Y_4, and Z. Y_1, Y_2, Y_3, and Y_4 are from the same scale so row-mean substution is used for all four variables. Note that these four items are not parallel because error variances are different.

```
> f <- rnorm(n, 0, 1)
> y1 <- 0.7*f + rnorm(n, 0, sqrt(0.51))
> y2 <- 0.7*f + rnorm(n, 0, sqrt(0.70))
> y3 <- 0.7*f + rnorm(n, 0, sqrt(0.90))
> y4 <- 0.7*f + rnorm(n, 0, sqrt(1))
> z <- rnorm(n, 0, 1)
> dat2 <- data.frame(y1, y2, y3, y4, z)
> dat2[as.logical(rbinom(n, 1, 0.1)), 1] <- NA
> dat2[as.logical(rbinom(n, 1, 0.2)), 2] <- NA
> dat2[as.logical(rbinom(n, 1, 0.05)), 3] <- NA
> dat2[as.logical(rbinom(n, 1, 0.1)), 4] <- NA
> dat2[as.logical(rbinom(n, 1, 0.05)), 5] <- NA
> FUN <- function(x) {
+         out <- mean(x[1:4], na.rm = TRUE)
+         if(is.na(out)) {
+                 return(x)
+         } else {
+                 impose <- is.na(x)
+                 impose[setdiff(1:length(x), 1:4)] <- FALSE
+                 x[impose] <- out
+                 return(x)
+         }
+ }
> dat2 <- t(apply(dat2, 1, FUN))
```

3.6 Full Information (Direct) Maximum Likelihood

In full information maximum likelihood (FIML), all available observations are used to estimate parameters. All missing observations are skipped during parameter estimation without imputation. The likelihood function is built from the likelihood values from each case and appropriately skips missing observations. Therefore, if all variables explaining missing data process are included in data analysis, this method can handle MAR. Full information maximum likelihood is the first good method introduced in this section. Although many packages in R can use FIML, in my opinion, `lavaan` is the easiest package. FIML is used when the `missing` argument is specified as `"ml"`.

```
> library(lavaan)
> script <- "
+   x3 ~~ x4 + z1 + z2
+   x4 ~~ z1 + z2
+   z1 ~~ z2
+ "
> out <- sem(script, data = dat, missing = "ml")
> summary(out, std = TRUE)

lavaan (0.5-18) converged normally after  29 iterations

  Number of observations                           100

  Number of missing patterns                        10

  Estimator                                         ML
  Minimum Function Test Statistic                0.000
  Degrees of freedom                                 0

Parameter estimates:

  Information                                 Observed
  Standard Errors                             Standard

                   Estimate  Std.err  Z-value  P(>|z|)   Std.lv  Std.nox
Covariances:
  x3 ~~
    x4                0.219    0.417    0.524    0.600    0.219    0.102
    z1                0.112    0.111    1.012    0.311    0.112    0.129
    z2               -0.081    0.120   -0.677    0.499   -0.081   -0.082
  x4 ~~
    z1               -0.083    0.336   -0.248    0.804   -0.083   -0.036
    z2               -0.190    0.427   -0.446    0.655   -0.190   -0.073
  z1 ~~
    z2               -0.009    0.120   -0.074    0.941   -0.009   -0.008
```

```
Intercepts:
    x3               -0.033    0.097   -0.341    0.733   -0.033   -0.037
    x4                3.475    0.322   10.794    0.000    3.475    1.460
    z1               -0.018    0.100   -0.179    0.858   -0.018   -0.018
    z2               -0.039    0.124   -0.318    0.750   -0.039   -0.036

Variances:
    x3                0.811    0.124                     0.811    1.000
    x4                5.666    1.076                     5.666    1.000
    z1                0.941    0.137                     0.941    1.000
    z2                1.194    0.190                     1.194    1.000
```

3.7 Multiple Imputation

In this method, missing observations are imputed using innformation from other variables in data set. However, this method does not simply put predicted values to the missing observations. It also adds random errors in the imputed values. Multiple imputed data sets are created to represent different sets of random errors. These random errors will affect different values of desired statistics across imputed data sets. Rubin's (1987) rule is used to pool parameter estimates and standard errors from each imputed data set. Let $\hat{Q}_j$ be an estimate of interest obtained from imputed data set j where $j = 1, 2, \ldots, m$ and m is the number of imputed data set. Let $\hat{U}_j$ be the standard error associated with $\hat{Q}_j$. Then, the overall estimate is the average of m estimates:

$$\bar{Q} = \frac{1}{m} \sum_{j=1}^{m} \hat{Q}_j \tag{13}$$

The standard error of the pooled estimate ($SE_{\bar{Q}}$) is composed of within-imputation standard error and between-imputation standard error.

$$SE_{\bar{Q}} = \sqrt{\bar{U} + \left(\frac{m+1}{m}\right) B} \tag{14}$$

where

$$\bar{U} = \frac{1}{m} \sum_{j=1}^{m} \hat{U}_j \tag{15}$$

and

$$\bar{B} = \frac{1}{m-1} \sum_{j=1}^{m} \left(\hat{Q}_j - \bar{Q}\right)^2 \tag{16}$$

Notice that both between-imputation and within-imputation variances are used to calculate the standard error. The uncertainty from missing observation is accounted for by the

between-imputation variance. That is, multiple imputation does not use imputed data as if they are observed. If all varibles explaining missing data process are included in multiple imputation, multiple imputation can handle MAR appropriately.

Many packages in R can run multiple imputation. This section will show the `mice` package. First, appropriate types of each variable. Nominal variable is specified by the `factor` function. Ordinal variable is specified by the `ordinal` function. Either `factor` or `ordered` function can be used for dichotomous variables. Then, the `mice` function is used to impute data where m is the number of imputation.

```
> library(mice)
> m <- 10
> newdat <- as.data.frame(dat)
> newdat[,"x1"] <- as.factor(newdat[,"x1"])
> newdat[,"x2"] <- ordered(newdat[,"x2"])
> newdat[,"x5"] <- ordered(newdat[,"x5"])
> imp <- mice(newdat, m = m, print = FALSE)
```

To pool statistical results across imputation, two methods can be used. First, if linear models are used, such as `lm`, `glm`, or `lmer`, pooling results can be done internally in the `mice` package by the `with` and `pool` function.

```
> fit <- with(imp, lm(x4 ~ x1 + x2 + x3 + x5))
> summary(pool(fit))
```

```
                   est        se           t       df   Pr(>|t|)      lo 95
(Intercept)  2.60285965 1.1822335  2.20164591 17.15453 0.04165672  0.1102754
x12          0.34882557 0.9792676  0.35621065 25.88840 0.72456945 -1.6645103
x13          0.64689048 0.9871360  0.65532051 18.92220 0.52014970 -1.4197841
x14          0.43163272 1.1761130  0.36699937 41.41087 0.71548829 -1.9428602
x22          0.90031274 0.9692762  0.92885053 33.27073 0.35965689 -1.0710852
x23          0.77712378 1.0151018  0.76556242 22.87321 0.45176468 -1.3234185
x24          0.08717752 1.4668809  0.05943054 26.74052 0.95305112 -2.9239805
x3          -0.16480694 0.5540247 -0.29747220 10.74001 0.77178084 -1.3878182
x52         -0.58747146 0.7526592 -0.78052787 23.20273 0.44297083 -2.1437131
                hi 95 nmis       fmi    lambda
(Intercept) 5.0954439   NA 0.5932916 0.5484864
x12         2.3621614   NA 0.4577138 0.4173777
x13         2.7135650   NA 0.5605239 0.5164046
x14         2.8061256   NA 0.3106295 0.2781203
x22         2.8717106   NA 0.3782527 0.3419683
x23         2.8776661   NA 0.4978904 0.4558257
x24         3.0983356   NA 0.4473118 0.4074648
x3          1.0582043   13 0.7474135 0.7043835
x52         0.9687702   NA 0.4932177 0.4513396
```

Note that X_1, X_2, and X_5 are changed to dummy variable where the first category is the baseline category.

If linear models are not used, users may explicitly impute data sets to completed data. Then, they can analyze each data set and Rubin's rule is implemented. For example, the results from independent t-test can be pooled.

```
> # Impute data explicitely
> dat.l <- NULL
> for(i in 1:m) dat.l[[i]] <- complete(imp, action = i)
> # Analyze each imputed data
> FUN <- function(x) {
+         result <- t.test(x4 ~ x5, data = x)
+         means <- result$estimate
+         diff <- means[1] - means[2]
+         se <- diff / result$statistic
+         c(diff, se)
+ }
> stat <- sapply(dat.l, FUN)
> # Implement Rubin's rule
> est <- mean(stat[1,])
> bvar <- var(stat[1,])
> wvar <- sum(stat[2,]^2)/m
> tvar <- wvar + ((m+1)/m)*bvar
> se <- sqrt(tvar)
> z <- est/se
> p <- 2 * (1 - pnorm(abs(z)))
```

The most dangerous thing in using `mice` is to not set the type of variables appropriately as the following example:

```
> wrongimp <- mice(dat, m = m, print = FALSE)
> fit <- with(wrongimp, lm(x4 ~ x1 + x2 + x3 + x5))
> summary(pool(fit))
                    est        se          t       df   Pr(>|t|)      lo 95
(Intercept)  2.92271974 1.2570453  2.3250712 24.35020 0.02871546  0.3302790
x1           0.08330515 0.3317332  0.2511210 28.92467 0.80349606 -0.5952421
x2           0.19630339 0.4020491  0.4882572 23.80571 0.62983494 -0.6338438
x3           0.25809751 0.4910474  0.5256061 11.89224 0.60882312 -0.8128790
x5          -0.49354045 0.5754381 -0.8576778 65.53659 0.39419616 -1.6425913
                hi 95 nmis       fmi    lambda
(Intercept) 5.5151605   NA 0.4835222 0.4427747
x1          0.7618524    9 0.4286755 0.3904913
x2          1.0264506   30 0.4908260 0.4497730
x3          1.3290740   13 0.7200188 0.6765847
x5          0.6555104    7 0.1771406 0.1524065
```

Notice that X_1, X_2, and X_5 are treated as continuous variable, which is not correct.

4 Debugging

Almost all the time, we cannot write a code without any problems or any bugs. In this section, some tips are introduced. Let's start with a function that has a problem.

```
> findmaxmi <- function(n) {
+   f <- rnorm(n, 0, 1)
+   y1star <- 0.5 + 0.9*f + rnorm(n, 0, 1)
+   y2star <- -0.5 + 0.7*f + rnorm(n, 0, 1)
+   y3star <- 0.2 + 0.5*f + rnorm(n, 0, 1)
+   y4star <- 0.3 + 0.3*f + rnorm(n, 0, 1)
+   y1 <- y1star > 0
+   y2 <- y2star > 0
+   y3 <- y3star > 0
+   y4 <- y4star > 0
+   dat <- data.frame(y1, y2, y3, y4)
+   script <- "f =~ y1 + y2 + y3 + y4"
+   out <- cfa(script, data = dat, ordered = c("y1", "y2", "y3", "y4"),
+     std.lv = TRUE)
+   return(which.max(inspect(out, "mi")$mi))
+ }
```

This function will create dichotomous data from one-factor CFA model with a given sample size. The created data are created by one-factor CFA and modification indices are extracted. The order of the maximum value of modification indices is saved. This function has a problem when n is low. The CFA model is not convergent so the modification indices cannot be computed as the following example:

```
> set.seed(123311)
> findmaxmi(50)

Error in lavTech(object, "inverted.information.expected") * nobs(object) :
  non-numeric argument to binary operator
In addition: Warning messages:
1: In lav_model_vcov(lavmodel = lavmodel, lavsamplestats = lavsamplestats,  :
  lavaan WARNING: could not compute standard errors!
  lavaan NOTE: this may be a symptom that the model is not identified.

2: In lav_model_test(lavmodel = lavmodel, lavpartable = lavpartable,  :
  lavaan WARNING: could not compute scaled test statistic

3: In lavaan::lavaan(model = script, data = dat, std.lv = TRUE, ordered = c("y1",  :
  lavaan WARNING: some estimated variances are negative
```

Let's pretend that we do not know this problem and find the bug in this problem. We can diagnose the problems by using the `browser()` or `traceback()` functions. First, the

`traceback()` function can be used immediately after finding the error. The results are the points of function that provide the errors.

```
> traceback()

7: modificationIndices(lavobject)
6: lavInspect(lavobject = object, what = what, add.labels = TRUE,
       add.class = TRUE, drop.list.single.group = TRUE)
5: .local(object, ...)
4: inspect(out, "mi")
3: inspect(out, "mi")
2: which.max(inspect(out, "mi")$mi) at #15
1: findmaxmi(50)
```

The `browser()` function can be added in a function so that, when we run a function, the running process stop at the position of `browser()`. That is, the `browser()` function is added in the second last line of the `findmaxmi` function above as follows:

```
  out <- cfa(script, data = dat, ordered = c("y1", "y2", "y3", "y4"),
    std.lv = TRUE)
  browser()
  return(which.max(inspect(out, "mi")$mi))
```

Let's run the same script again. The browse line appears:

```
> set.seed(123311)
> findmaxmi(50)

Called from: findmaxmi(50)
Browse[1]>
```

From here, we can check the objects inside the functions whether they are correct and check whether each line provide appropriate results. Here we will see that the modification indices cannot be computed.

```
Browse[1]> inspect(out, "mi")

Error in lavTech(object, "inverted.information.expected") * nobs(object) :
  non-numeric argument to binary operator
```

We can use `Q` to kill the process or `c` to continue evaluate the function.

```
Browse[1]> Q

>
```

From here, we would like to check whether the modification indices can be computed. If not, we should have a way to not let the error break the function. Otherwise, our simulation cannot be run.

We can use `try()` function to crop a line of command. If the line of command cropped by the `try()` function breaks, the process will skip the line and continue running the function. For example, we can write a following command:

```
> addition <- function(a, b) {
+         result <- NA
+         try(result <- a + b)
+         result
+ }
> addition("skill", "effort")

[1] NA
```

Although strings cannot be added together, using `try` allows the function to return `NA` without breaking the process. Actually, the result of the `try` function can be saved. If errors occur, the result of the `try` function is saved as a `"try-error"` object so we can check whether the errors occur in the line of command inside the `try` function. For example, the `addition` function can be modified:

```
> addition <- function(a, b) {
+   result <- NA
+   tryout <- try(result <- a + b)
+   if(is(tryout, "try-error")) {
+     print("Two things cannot be added.")
+     return(NA)
+   } else {
+     return(result)
+   }
+ }
> addition("skill", "effort")

[1] "Two things cannot be added."
[1] NA
```

Therefore, the last line of the `findmaxmi` function above can be fixed as follows:

```
+   tryout <- try(result <- return(which.max(inspect(out, "mi")$mi)))
+   if(is(tryout, "try-error")) {
+     print("Cannot calculate the modification indices.")
+     return(NA)
+   } else {
+     return(result)
+   }
```

Alternatively, we can use the `tryCatch` function. If the `tryCatch` function got an error, it will evaluate the function provided in the `error` argument. For example, the `addition` function can be modified to provide the error message.

```
> addition <- function(a, b) {
+   result <- NA
+   tryCatch(result <- a + b, error = function(e)
+     print("Two things cannot be added."))
+   result
+ }
> addition("skill", "effort")

[1] "Two things cannot be added."
[1] NA
```

Thus, the last line of the `findmaxmi` function can be also modified as follows:

```
+   result <- NA
+   tryCatch(result <- return(which.max(inspect(out, "mi")$mi)),
+     error = function(e) print("Cannot calculate modification indices"))
+   return(result)
```

If we would make a function for other people to use, it is very helpful to check whether users provide valid arguments. For example, in the `addition` function, the `a` and `b` arguments can be checked whether they are numeric. This check could be done easily using the if statement. Two functions, `warning` and `stop`, can be used if users provide invalid arguments. The `warning` function is used to provide a warning message and the function still continues to run. The `stop` funciton, however, is used to provide an error message and stops the function immediately. For example, we can use the `warning` function in `addition`:

```
> addition <- function(a, b) {
+   if(!is.numeric(a)) warning("a is not numeric")
+   if(!is.numeric(b)) warning("b is not numeric")
+   result <- NA
+   tryCatch(result <- a + b, error = function(e)
+     print("Two things cannot be added together."))
+   result
+ }
> addition("skill", "effort")

[1] "Two things cannot be added together."
[1] NA
```

Alternatively, we may use the `stop` function:

```
> addition <- function(a, b) {
+   if(!is.numeric(a)) stop("a is not numeric")
+   if(!is.numeric(b)) stop("b is not numeric")
+   result <- NA
+   tryCatch(result <- a + b, error = function(e)
+     print("Two things cannot be added together."))
+   result
+ }
> addition("skill", "effort")

Error in addition("skill", "effort") : a is not numeric
```

Thus, in the `findmaxmi` function, we may provide a warning message if sample size is low (says less than 100).

```
> findmaxmi <- function(n) {
+   if(n < 100) warning("n might be too low for categorical CFA.")
+   f <- rnorm(n, 0, 1)
```

5 Simulation Example

Let's run a simulation to investigate the impact of missing data on simple regression. The model is $Y = \alpha + \beta X + e$, where $X \sim N(0, 1)$, $e \sim N(0, \sqrt{0.75})$, $\alpha = 0$, and $\beta = 0.5$. The sample size is 200. Two missing patterns are investigated: MCAR and MAR where X is the only variable that has missing values. The proportion of missing values can be 5%, 10%, 20%, or 40%. In MAR, let's Z be a normally distributed variable with the mean of 0 and the variance of 1. The correlation between X and Z is .5. The missing value of X depends on Z by logistic regression model. The logistic regression coefficient of Z is 0.5 in determining the missing values. The intercepts are computed such that the expected missing values are 5%, 10%, or 20% when $Z = 0$. The generated data will be analyzed by two methods: listwise deletion and multiple imputation with Z included. The number of imputation is 5. The slopes obtained from two methods are compared in terms of the estimates and standard errors. The data generating script is as follows:

```
> pm <- 0.05 # Can be .05, .1, .2, or .4
> mp <- 1 # 1 = MCAR, 2 = MAR
> n <- 200
> library(MASS)
> S <- matrix(c(1, 0.5, 0.5, 1), 2, 2)
> XS <- mvrnorm(n, rep(0, 2), S)
> x <- XS[,1]
> y <- 0.5 * x + rnorm(n, 0, sqrt(0.75))
> z <- XS[,2]
> if(mp == 1) {
+   mx <- as.logical(rbinom(n, 1, pm))
```

```
+ } else {
+   mhat <- log(pm/(1 - pm)) + 0.5 * z
+   phat <- 1 / (1 + exp(-mhat))
+   mx <- as.logical(rbinom(n, 1, phat))
+ }
> x[mx] <- NA
> dat <- data.frame(x, y, z)
```

Note that the intercept can be computed by simply substitute Z as 0, which is the population mean (expected value) of Z, in logistic regression:

$$\log\left(\frac{\Pr\left(M_X = 1\right)}{1 - \Pr\left(M_X = 1\right)}\right) = \alpha + 0.5 \cdot (0) \tag{17}$$

The data analysis for listwise deletion can be implemented:

```
> summary(lm(y ~ x, data = dat))[["coefficients"]][2, 1:2]

  Estimate Std. Error
0.56677081 0.05885496
```

The data analysis using multiple imputation can be implemented:

```
> library(mice)
> imps <- mice(dat, m = 5, print = FALSE)
> poolthing <- with(imps, lm(y ~ x))
> summary(pool(poolthing))[2, 1:2]

      est        se
0.5651583 0.0586399
```

The whole simulation can be run as follows:

```
> library(mice)
> set.seed(123321)
> pms <- c(0.05, 0.1, 0.2, 0.4)
> mps <- c(1, 2) # MCAR and MAR
> n <- 200
> nrep <- 1000
> S <- matrix(c(1, 0.5, 0.5, 1), 2, 2)
> ovresult <- NULL
> for(k in 1:length(pms)) {
+   pm <- pms[k]
+   for(j in 1:length(mps)) {
+     mp <- mps[j]
+     result <- matrix(NA, nrep, 6)
+     for(i in 1:nrep) {
```

```
+       XS <- mvrnorm(n, rep(0, 2), S)
+       x <- XS[,1]
+       y <- 0.5 * x + rnorm(n, 0, sqrt(0.75))
+       z <- XS[,2]
+       if(mp == 1) {
+         mx <- as.logical(rbinom(n, 1, pm))
+       } else {
+         mhat <- log(pm/(1 - pm)) + 0.5 * z
+         phat <- 1 / (1 + exp(-mhat))
+         mx <- as.logical(rbinom(n, 1, phat))
+       }
+           compdat <- data.frame(x, y)
+           result[i, 1:2] <-
+                 summary(lm(y ~ x, data = compdat))[["coefficients"]][2, 1:2]
+       x[mx] <- NA
+       dat <- data.frame(x, y, z)
+       # Listwise
+       result[i, 3:4] <-
+                 summary(lm(y ~ x, data = dat))[["coefficients"]][2, 1:2]
+       # mice, MI
+       imps <- mice(dat, m = 5, print = FALSE)
+       poolthing <- with(imps, lm(y ~ x))
+       result[i, 5:6] <- summary(pool(poolthing))[2, 1:2]
+     }
+     result <- cbind(pm, mp, result)
+     ovresult <- rbind(ovresult, result)
+   }
+ }
> colnames(ovresult) <- c("pm", "mp", "b", "seb", "blist", "seblist",
+         "bmi", "sebmi")
```

In this simulation, the results of completed data are also saved. This is helpful in comparing two methods of missing data handling. First, the relative biases in parameter estimates are investigated. The biases can be defined in two ways: (a) comparing with population values and (b) comparing with the results from completed data.

```
> ave <- aggregate(cbind(b, seb, blist, seblist, bmi, sebmi) ~ pm + mp,
+         data = ovresult, FUN = mean)
> empse <- aggregate(cbind(b, blist, bmi) ~ pm + mp, data = ovresult, FUN = sd)
> conds <- ave[,1:2]
> relbiasest <- cbind(conds, blist = (ave[,"blist"] - 0.5)/0.5,
+         bmi = (ave[,"bmi"] - 0.5)/0.5)
> relbiasest2 <- cbind(conds, blist = (ave[,"blist"] - ave[,"b"])/ave[,"b"],
+         bmi = (ave[,"bmi"] - ave[,"b"])/ave[,"b"])
> round(relbiasest, 4)
```

```
    pm mp   blist     bmi
1 0.05  1  0.0024  0.0018
2 0.10  1  0.0040  0.0028
3 0.20  1  0.0019  0.0008
4 0.40  1  0.0000 -0.0091
5 0.05  2 -0.0004 -0.0011
6 0.10  2  0.0032  0.0022
7 0.20  2  0.0051  0.0010
8 0.40  2 -0.0058 -0.0171
```

```
> round(relbiasest2, 4)
```

```
    pm mp   blist     bmi
1 0.05  1  0.0005 -0.0001
2 0.10  1  0.0006 -0.0006
3 0.20  1 -0.0012 -0.0023
4 0.40  1  0.0023 -0.0068
5 0.05  2  0.0000 -0.0008
6 0.10  2  0.0002 -0.0008
7 0.20  2 -0.0009 -0.0050
8 0.40  2  0.0014 -0.0100
```

Multiple imputation provides higher biases in parameter estimates than the listwise deletion when the proportion of missing values is high (e.g., .40). However, all biases are still negligible because the magnitudes are less than 0.05.

Second, the relative biases in standard errors are investigated. Again, the biases can be defined in two ways. The averaged stanadard errors can be compared with the empirical standard errors obtained from the estimates from the missing data handling methods. These values can be interpreted as whether the calculated standard errors could estimate the standard error in population correctly. Alternatively, the averaged standard errors can be compared with the empirical standard errors obtained from complete data. These values can be interpreted as the recovery of missing information compared to complete data.

```
> relbiasse <- cbind(conds,
+         blist = (ave[,"seblist"] - empse[,"blist"])/empse[,"blist"],
+         bmi = (ave[,"sebmi"] - empse[,"bmi"])/empse[,"bmi"])
> relbiasse2 <- cbind(conds,
+         blist = (ave[,"seblist"] - empse[,"b"])/empse[,"b"],
+         bmi = (ave[,"sebmi"] - empse[,"b"])/empse[,"b"])
> round(relbiasse, 4)
```

```
    pm mp   blist     bmi
1 0.05  1 -0.0435 -0.0496
2 0.10  1  0.0109  0.0046
3 0.20  1 -0.0127 -0.0239
4 0.40  1 -0.0064 -0.0505
```

```
5 0.05  2 -0.0308 -0.0343
6 0.10  2 -0.0034 -0.0008
7 0.20  2 -0.0219 -0.0377
8 0.40  2  0.0232 -0.0053
```

```
> round(relbiasse2, 4)
```

```
    pm mp   blist     bmi
1 0.05  1 -0.0230 -0.0307
2 0.10  1  0.0420  0.0262
3 0.20  1  0.1064  0.0661
4 0.40  1  0.2359  0.1208
5 0.05  2 -0.0064 -0.0152
6 0.10  2  0.0641  0.0428
7 0.20  2  0.1167  0.0677
8 0.40  2  0.3306  0.2126
```

For the first type of relative bias in standard errors, there are not obvious patterns in comparing both methods. However, all methods provide negligible biases in standard errors because the values are lower than .1. Regarding to the second type of relative bias in standard errors, the listwise deletion is better than multiple imputation in the condition of 5% missing. If the proportion of missing is greater than 5%, multiple imputation provides the amount of standard errors closer to the empirical standard error from complete data.

Finally, sometimes, especially when sample size is low, there is no missing value in the data at all. That is, `mx` is all `FALSE`. We might want to write a loop such that `mx` must provide at least one missing value (one `TRUE`). In this case, we can use while loop. For example,

```
> set.seed(123321)
> n <- 3
> mx <- rep(FALSE, 3)
> while(all(!mx)) {
+         mx <- as.logical(rbinom(n, 1, 0.05))
+         print(mx)
+ }
```

```
[1] FALSE FALSE FALSE
[1] FALSE FALSE FALSE
[1] FALSE FALSE FALSE
[1] FALSE FALSE FALSE
[1] FALSE FALSE FALSE
[1] FALSE FALSE FALSE
[1] FALSE FALSE  TRUE
```

The while statement evaluates the statement inside the parenthesis. If the statement in parenthesis is still `TRUE` (no missing value), then the while statement will run the statements inside the curly bracket until the statement in the parenthesis is `FALSE` (some missing values). Therefore, we could change the code that generates missing data above as

```
+       mx <- rep(FALSE, n)
+       while(all(!mx)) {
+         if(mp == 1) {
+           mx <- as.logical(rbinom(n, 1, pm))
+         } else {
+           mhat <- log(pm/(1 - pm)) + 0.5 * z
+           phat <- 1 / (1 + exp(-mhat))
+           mx <- as.logical(rbinom(n, 1, phat))
+         }
+       }
```

6 Exercises

Let's run a simulation to investigate the impact of missing data on independent t-test with equal variances assumed. Let Y_1 and Y_2 be the scores for each group. $Y_1 \sim N(50, 10)$ and $Y_2 \sim N(45, 10)$. The sample size for each group is 50. Two missing patterns are investigated: MCAR and MAR where Y is the only variable that has missing values. The proportion of missing values can be 5%, 10%, 20%, or 40%. In MAR, let's $Z = D_X + e_Z$ where $e_Z \sim N(0, 1)$ and $D_X = 1, 0$ for Groups 1 and 2, respectively. The missing value of Y depends on Z by logistic regression model. The logistic regression coefficient of Z is 0.5 in determining the missing values. The intercepts are computed such that the expected missing values are 5%, 10%, or 20% when $Z = 0$. Note that the population mean of Z is 0.5. The generated data will be analyzed by two methods: listwise deletion and multiple imputation with Z included. The number of imputation is 5. The mean differences obtained from two methods are compared in terms of the estimates and standard errors.