Prediction, Imputation, & Cross-Validation

Statistics & Methodology Lecture 6

Kyle M. Lang

Department of Methodology & Statistics
Tilburg University

Outline

1. Prediction
2. Cross-validation
3. Missing data imputation

Prediction

So far, we've focused primarily on inferences about the estimated regression coefficients.

- Asking questions about how $\mathbf{X}$ is related to Y

We can also use linear regression for *prediction*.

- Given a new observation, X_m, what outcome value, $\hat{Y}_m$, does our model attribute to the mth observation?

Prediction

Train a model to predict employee performance using features extracted from CVs.

- When screening applicants for a new position, use the data in their CVs to predict their expected performance.

Predict recidivism risk based on personal history, criminal history, and in-prison behavior record.

- When evaluating a parole application, calculate the predicted chance of recidivism.

Predict future gasoline prices based on geo-political events in oil-producing countries.

- If conflict escalates in the Middle East, adjust the appropriate features and project likely changes in gasoline prices.

Prediction Example

To fix ideas, let's reconsider the *diabetes* data and the following model:

$$Y_{LDL} = \beta_0 + \beta_1 X_{BP} + \beta_2 X_{gluc} + \beta_3 X_{BMI} + \varepsilon$$

Training this model on the first $N = 400$ patients' data produces the following fitted model:

$$\hat{Y}_{LDL} = 22.135 + 0.089 X_{BP} + 0.498 X_{gluc} + 1.48 X_{BMI}$$

Prediction Example

To fix ideas, let's reconsider the *diabetes* data and the following model:

$$Y_{LDL} = \beta_0 + \beta_1 X_{BP} + \beta_2 X_{gluc} + \beta_3 X_{BMI} + \varepsilon$$

Training this model on the first $N = 400$ patients' data produces the following fitted model:

$$\hat{Y}_{LDL} = 22.135 + 0.089 X_{BP} + 0.498 X_{gluc} + 1.48 X_{BMI}$$

Suppose a new patient presents with $BP = 121$, $gluc = 89$, and $BMI = 30.6$. We can predict their LDL score by:

$$\begin{aligned} \hat{Y}_{LDL} &= 22.135 + 0.089(121) + 0.498(89) + 1.48(30.6) \\ &= 122.463 \end{aligned}$$

Prediction with Centered X Variables

Suppose we fit the preceding model with BP centered at 90, $gluc$ centered at 100, and BMI centered at 30.

- We'd get the following fitted model:

$$\hat{Y}_{LDL} = 124.289 + 0.089X_{BP.90} + 0.498X_{gluc.100} + 1.48X_{BMI.30}$$

Prediction with Centered X Variables

Suppose we fit the preceding model with BP centered at 90, $gluc$ centered at 100, and BMI centered at 30.

- We'd get the following fitted model:

$$\hat{Y}_{LDL} = 124.289 + 0.089X_{BP.90} + 0.498X_{gluc.100} + 1.48X_{BMI.30}$$

Now, let's generate predictions for our patient with $BP = 121$, $gluc = 89$, and $BMI = 30.6$:

$$\begin{aligned}\hat{Y}_{LDL} &= 124.289 + 0.089(121) + 0.498(89) + 1.48(30.6) \\ &= 224.617\end{aligned}$$

Prediction with Centered X Variables

Suppose we fit the preceding model with BP centered at 90, $gluc$ centered at 100, and BMI centered at 30.

- We'd get the following fitted model:

$$\hat{Y}_{LDL} = 124.289 + 0.089X_{BP.90} + 0.498X_{gluc.100} + 1.48X_{BMI.30}$$

Now, let's generate predictions for our patient with $BP = 121$, $gluc = 89$, and $BMI = 30.6$:

$$\begin{aligned}\hat{Y}_{LDL} &= 124.289 + 0.089(121) + 0.498(89) + 1.48(30.6) \\ &= 224.617\end{aligned}$$

To get the correct prediction, we need to plug-in the centered X values:

$$\begin{aligned}\hat{Y}_{LDL} &= 124.289 + 0.089(121 - 90) + 0.498(89 - 100) + 1.48(30.6 - 30) \\ &= 122.463\end{aligned}$$

Interval Estimates for Prediction

To quantify uncertainty in our predictions, we want to use an appropriate interval estimate.

- Two flavors of interval are applicable to predictions:
  1. Confidence intervals for $\hat{Y}$
  2. Prediction intervals for a specific observation

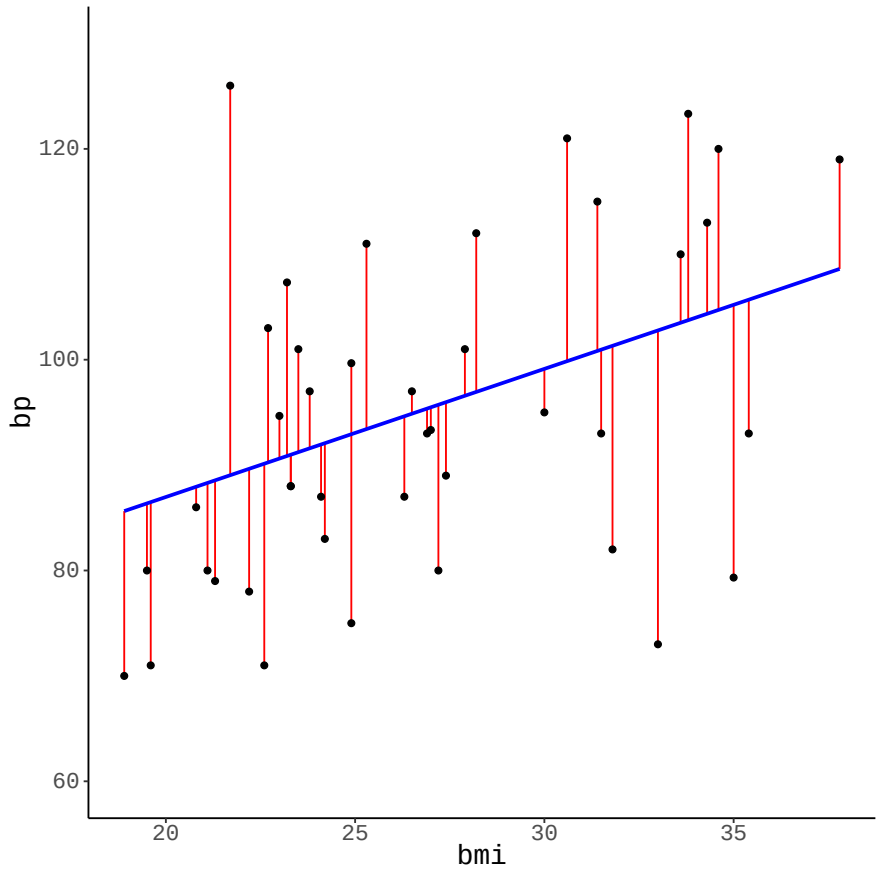
- CIs for $\hat{Y}$ give a likely range (in the sense of coverage probability and "confidence") for the true conditional mean of Y, $\mu_{Y|X^*}$.
  - They only account for uncertainty in the estimated regression coefficients, $\{\hat{\beta}_0, \hat{\beta}_p\}$.

- Prediction intervals give a likely range (in the same sense as CIs) for future outcome values, Y^*.
  - They also account for the regression errors, ε.

Confidence vs. Prediction Intervals

Let's visualize the predictions from a simple model:

$$Y_{BP} = \hat{\beta}_0 + \hat{\beta}_1 X_{BMI} + \hat{\varepsilon}$$

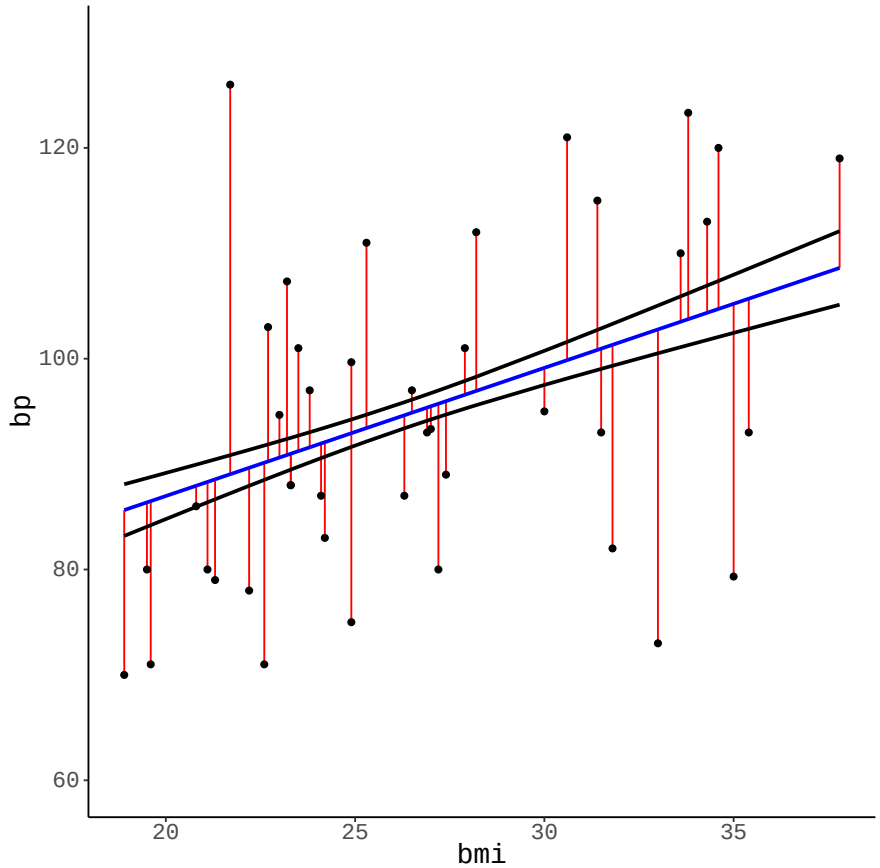


Confidence vs. Prediction Intervals

Let's visualize the predictions from a simple model:

$$Y_{BP} = \hat{\beta}_0 + \hat{\beta}_1 X_{BMI} + \hat{\varepsilon}$$

- CIs for $\hat{Y}$ ignore the errors, ε.
  - They only care about the best-fit line, $\beta_0 + \beta_1 X_{BMI}$.

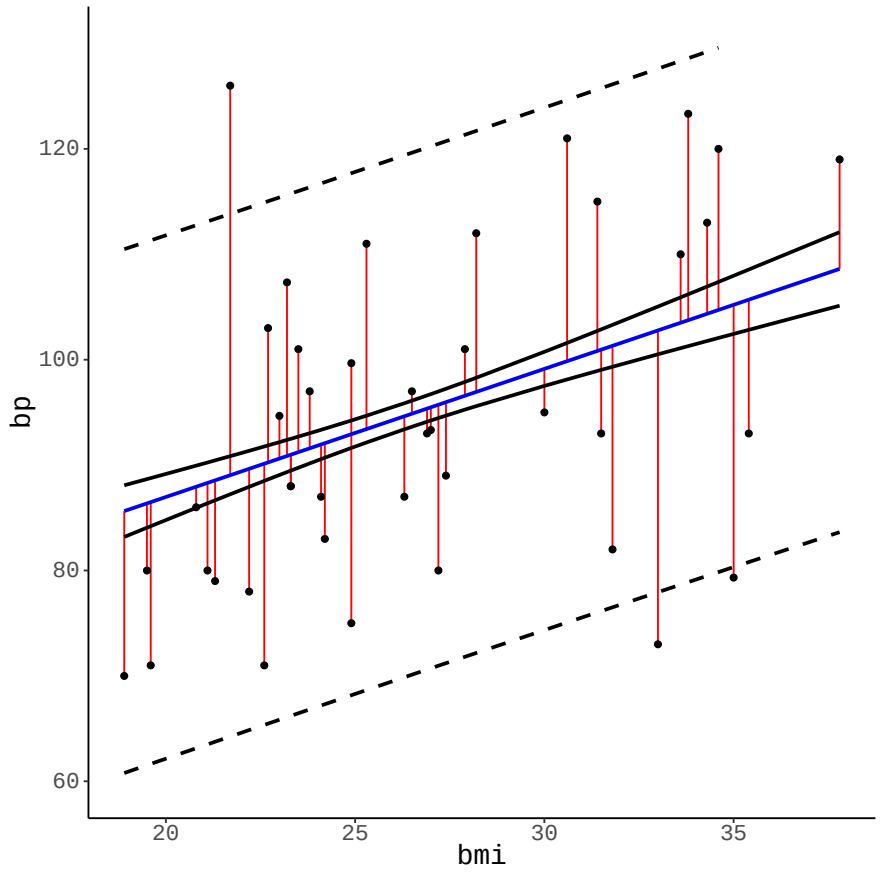


Confidence vs. Prediction Intervals

Let's visualize the predictions from a simple model:

$$Y_{BP} = \hat{\beta}_0 + \hat{\beta}_1 X_{BMI} + \hat{\varepsilon}$$

- CIs for $\hat{Y}$ ignore the errors, ε.
  - They only care about the best-fit line, $\beta_0 + \beta_1 X_{BMI}$.
- Prediction intervals are wider than CIs.
  - They account for the additional uncertainty contributed by ε.



Confidence vs. Prediction Intervals

Assume we want predictions for a new observation, X^*. Then our intervals can be computed as follows:

- Confidence interval:

$$CI = \hat{Y} \pm t_{crit} \sqrt{\frac{\hat{\sigma}^2}{N} + \frac{\left(X^* - \bar{X}\right)^2}{\left(\sum_{n=1}^{N} X_n - \bar{X}\right)^2}}$$

- Prediction interval:

$$PI = \hat{Y} \pm t_{crit} \sqrt{\hat{\sigma}^2 + \frac{\hat{\sigma}^2}{N} + \frac{\left(X^* - \bar{X}\right)^2}{\left(\sum_{n=1}^{N} X_n - \bar{X}\right)^2}}$$

Interval Estimates Example

Going back to our hypothetical "new" patient, we get the following 95% interval estimates:

$$95\%CI_{\hat{Y}_{LDL}} = [115.599; 129.327]$$

$$95\%PI = [66.559; 178.368]$$

- We can be 95% confident that the average LDL of patients with $BP = 121$, $gluc = 89$, and $BMI = 30.6$ will be somewhere between 115.599 and 129.327.
- We can be 95% confident that the LDL of a specific patient with $BP = 121$, $gluc = 89$, and $BMI = 30.6$ will be somewhere between 66.559 and 178.368.

Specifying Predictive Models

When focused on inferences about regression coefficients, we care very much about the predictors entered into the model.

- Partial regression coefficients must be interpreted as controlling for all other predictors.

Specifying Predictive Models

When focused on inferences about regression coefficients, we care very much about the predictors entered into the model.

- Partial regression coefficients must be interpreted as controlling for all other predictors.

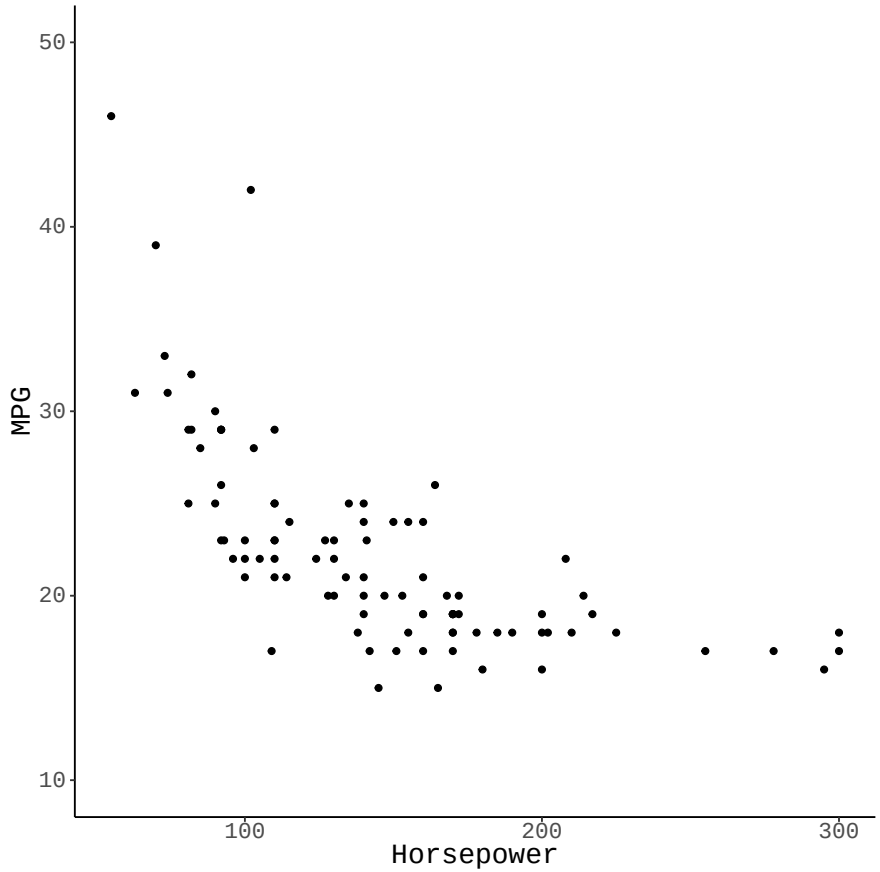
When focused on prediction, we often don't care as much about the specific variables that enter the model.

- We prefer whatever set of features produces the best predictive performance.
- We may want to know which are the "best" predictors.
  - We usually want the data to "give" us this answer.

Aside: Polynomial Regression

We may hypothesize a curvilinear relationship between X and Y.

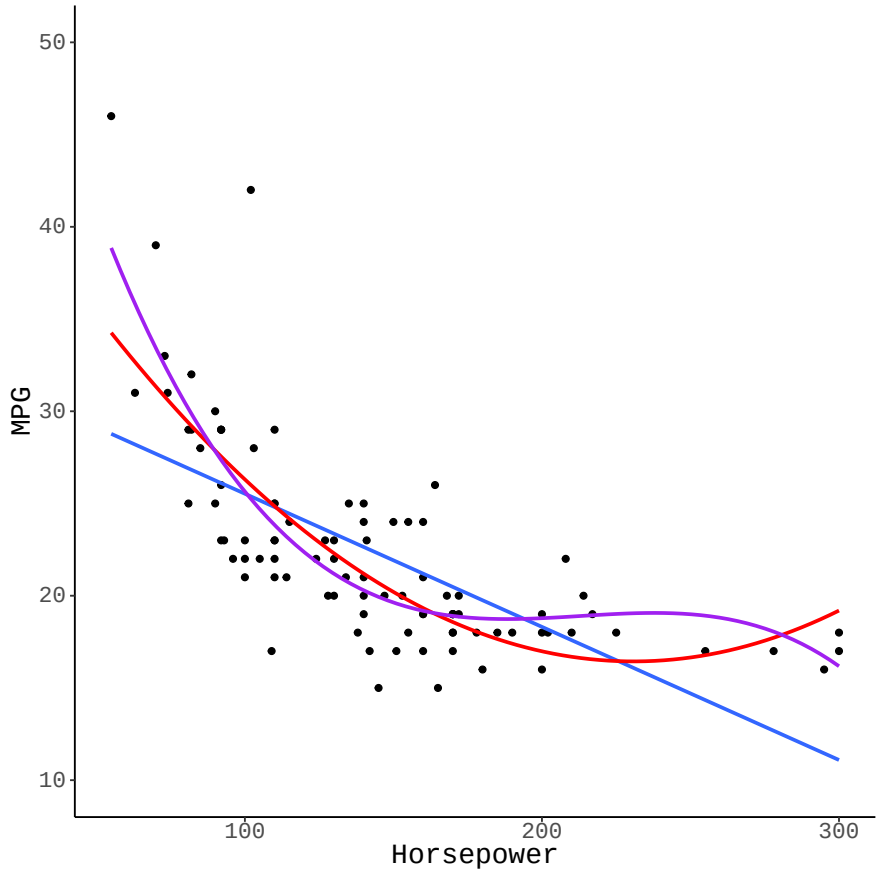
- Polynomial regression adds powered transformations of the predictors into the model.
- Polynomial terms (i.e., power terms) model curvature in the relationships.



Aside: Polynomial Regression

Polynomials are one way to model curvilinear relationships.

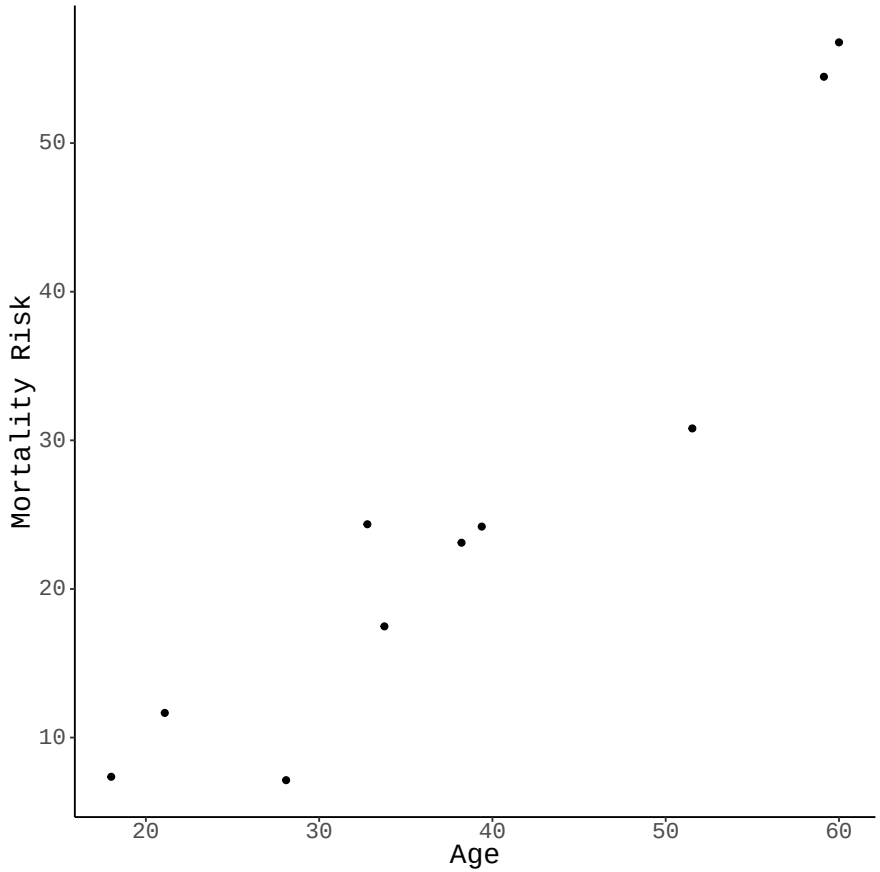
- $\hat{Y}_{mpg} = \hat{\beta}_0 + \hat{\beta}_1 X_{hp}$
- $\hat{Y}_{mpg} = \hat{\beta}_0 + \hat{\beta}_1 X_{hp} + \hat{\beta}_2 X_{hp}^2$
- $\hat{Y}_{mpg} = \hat{\beta}_0 + \hat{\beta}_1 X_{hp} + \hat{\beta}_2 X_{hp}^2 + \hat{\beta}_3 X_{hp}^3$



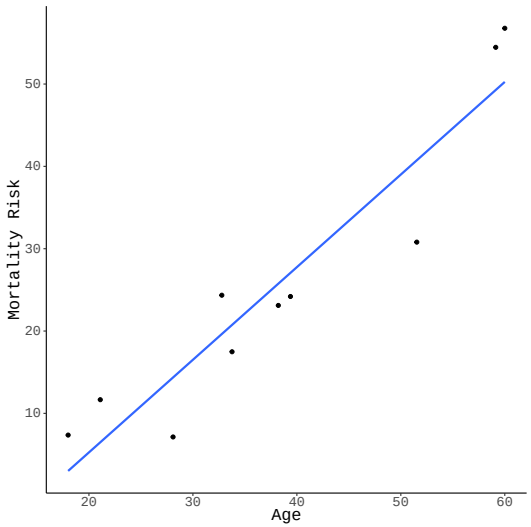
Evaluating Predictive Performance

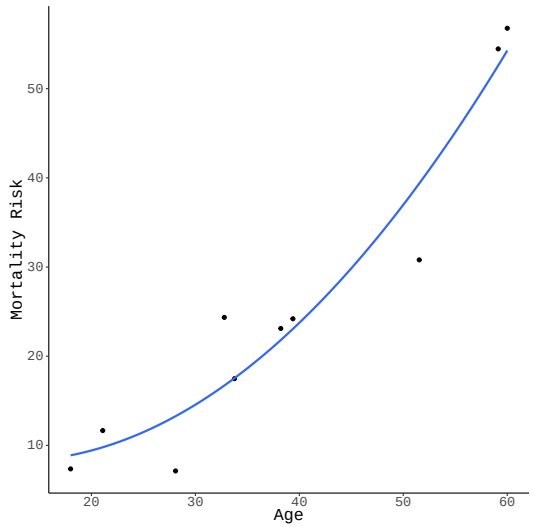
How do we assess "good" prediction?

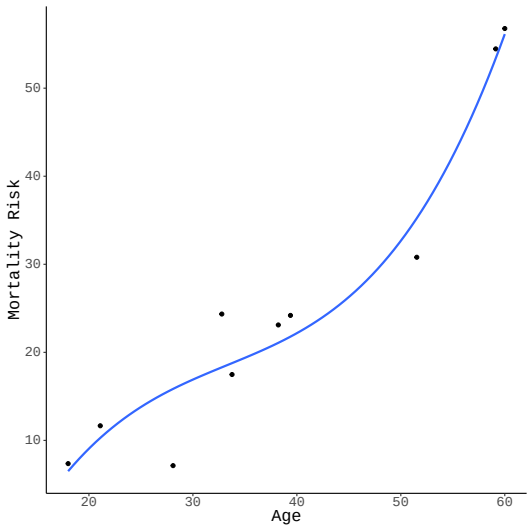
- Can we simply find the model that best predicts the data used to train the model?
- What are we trying to do when building a predictive model?
- Can we quantify this objective with some type of fit measure?

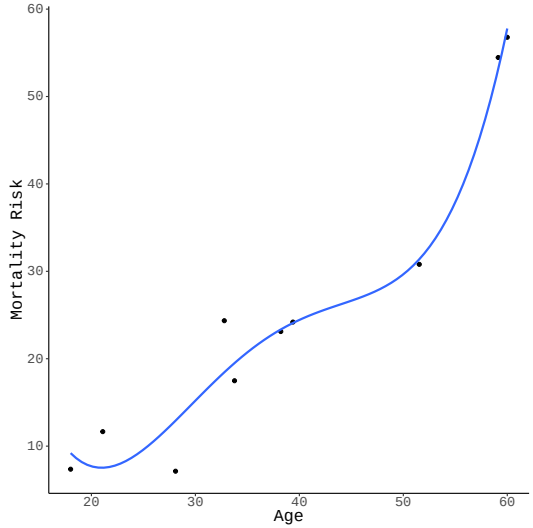


Different Possible Models





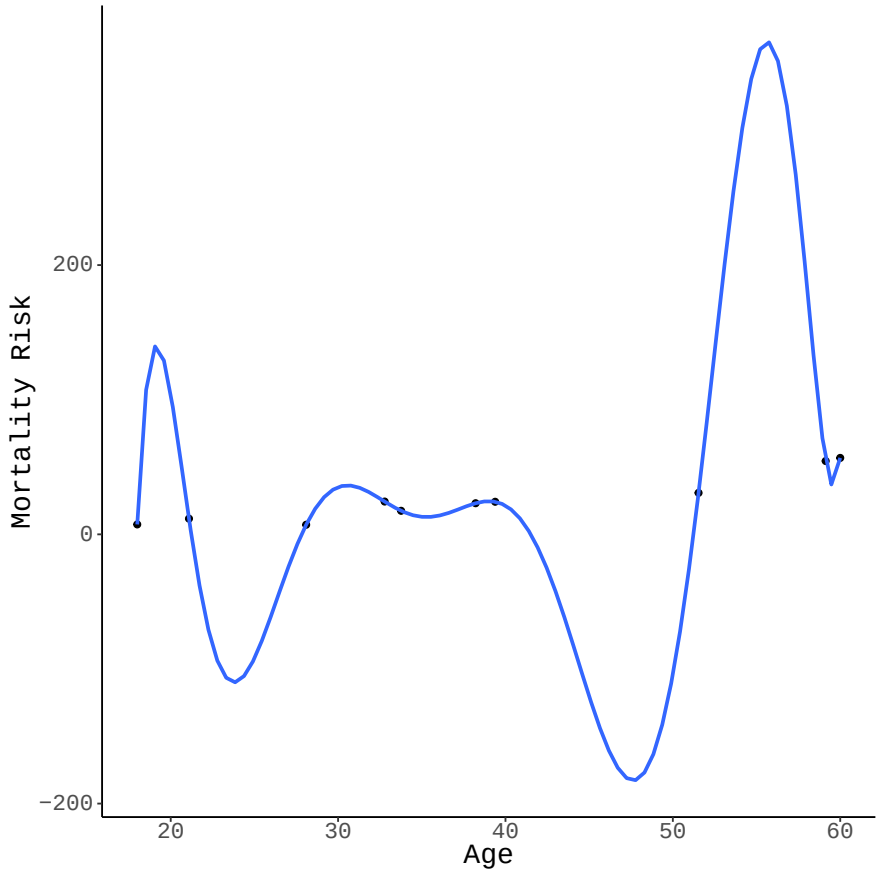




Over-fitting

We can easily go too far.

- Enough polynomial terms will exactly replicate any data.
- Is this what we're trying to do?
- What kind of issues arise in the extreme case?



Consequences of Over-fitting

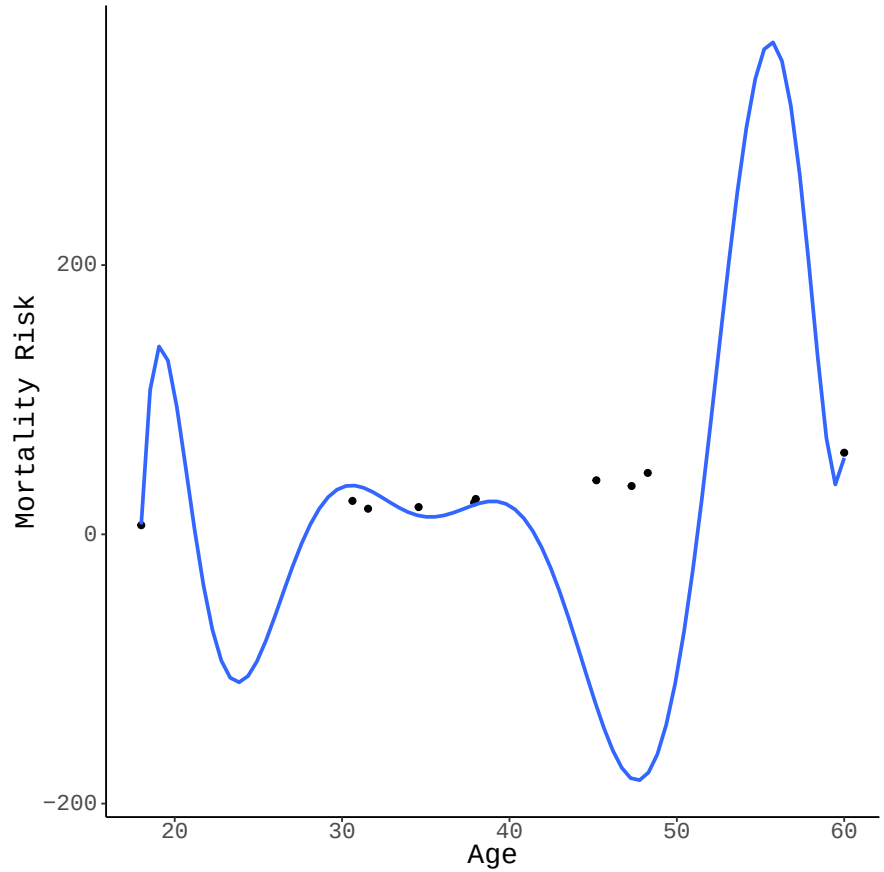
Should we be pleased to be able to perfectly predict mortality risk?

- Is our model useful?
- What happens if we try to apply our fitted model to new data?

Consequences of Over-fitting

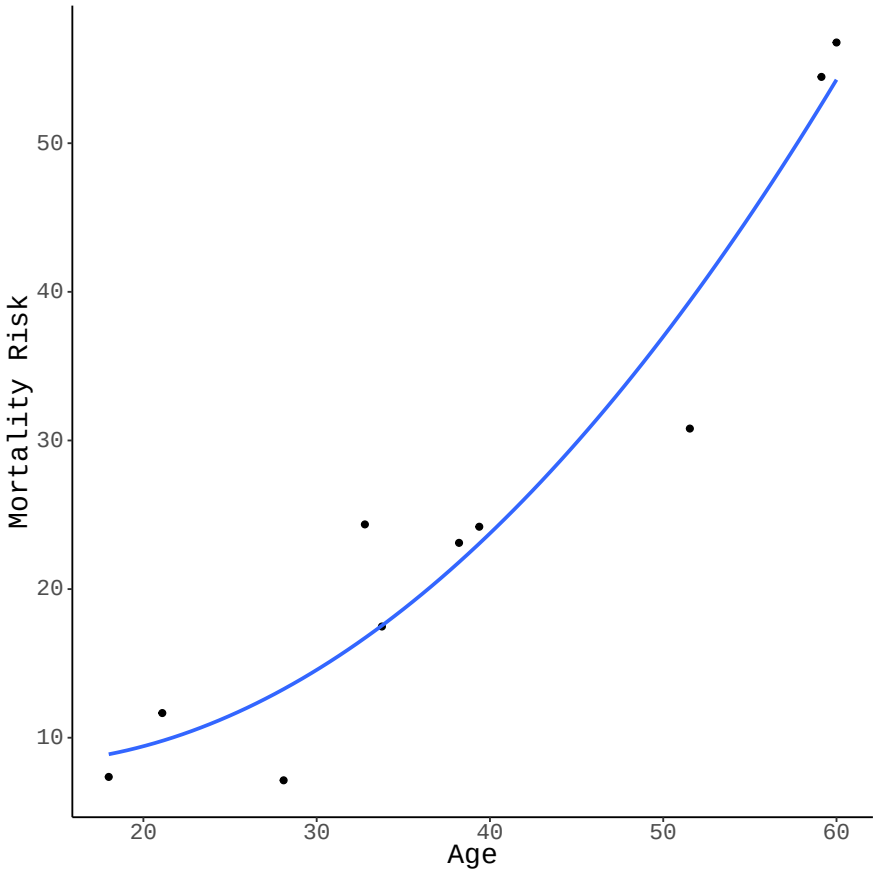
Should we be pleased to be able to perfectly predict mortality risk?

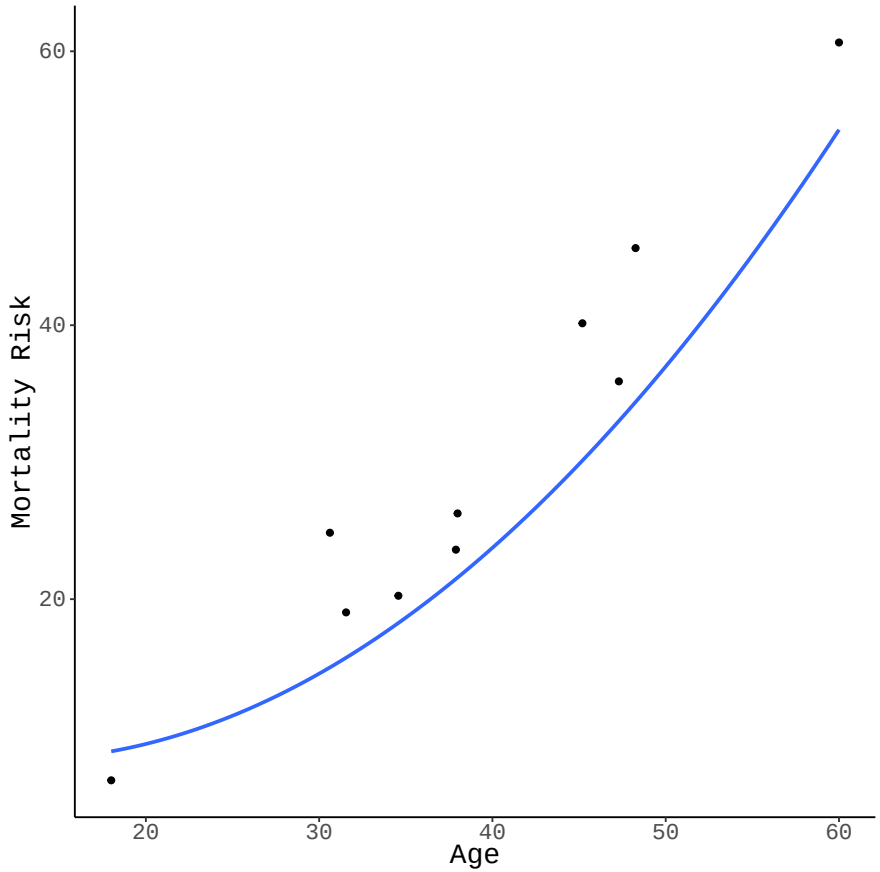
- Is our model useful?
- What happens if we try to apply our fitted model to new data?



Correct Fit

Let's try something a bit more reasonable.





A Sensible Goal

Our goal is to train a model that can best predict *new data*.

- The predictive performance on the training data is immaterial.
- We can always fit the training data arbitrarily well.
- Fit to the training data will always be at-odds with fit to future data.

This conflict is the driving force behind the *bias-variance trade-off*

Model Fit for Prediction

When assessing predictive performance, we will most often use the *mean squared error* (MSE) as our criterion.

$$
\begin{aligned}
MSE &= \frac{1}{N}\sum_{n=1}^{N}\left(Y_n - \hat{Y}_n\right)^2 \\
&= \frac{1}{N}\sum_{n=1}^{N}\left(Y_n - \hat{\beta}_0 - \sum_{p=1}^{P}\hat{\beta}_p X_{np}\right)^2 \\
&= \frac{RSS}{N}
\end{aligned}
$$

Training vs. Test MSE

The MSE on the preceding slide (i.e., the only MSE we've considered, so far) is computing entirely from training data.

- *Training MSE*

What we want is a measure of fit to new, *testing* data.

- *Test MSE*
- Given M new observations $\{Y_m, X_{m1}, X_{m2}, \ldots, X_{mP}\}$, and a fitted regression model, $f(\mathbf{X})$, defined by the coefficients $\{\hat{\beta}_0, \hat{\beta}_1, \hat{\beta}_2, \ldots, \hat{\beta}_P\}$, the *Test MSE* is given by:

$$MSE = \frac{1}{M}\sum_{m=1}^{M}\left(Y_m - \hat{\beta}_0 - \sum_{p=1}^{P}\hat{\beta}_p X_{mp}\right)^2$$

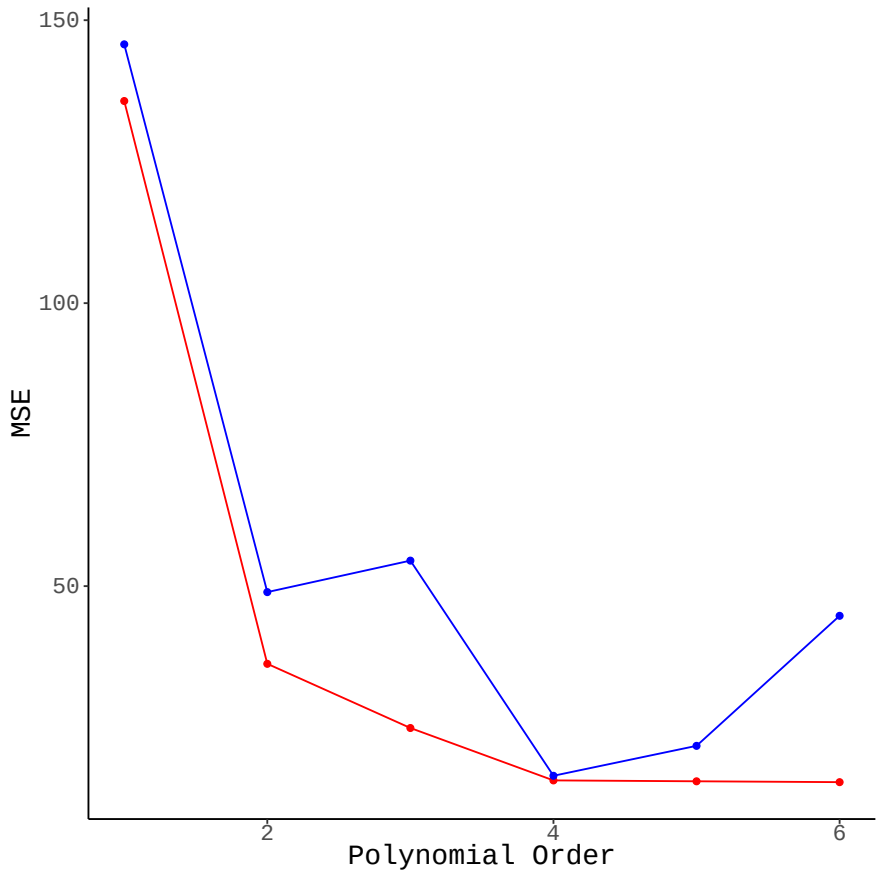
Training vs. Test MSE

Training MSE will always decrease in response to increased model complexity.

- Note the red line in the plot.

Test MSE will reach a minimum at some "optimal" level of model complexity.

- Further complicating the model will increase Test MSE.
- Note the blue line.



Training vs. Test MSE

In the last lecture, we compared the following models:

$$Y_{BP} = \beta_0 + \beta_1 X_{age} + \beta_2 X_{LDL} + \beta_3 X_{HDL} + \beta_4 X_{BMI} + \varepsilon \quad (1)$$

$$Y_{BP} = \beta_0 + \beta_1 X_{age} + \beta_2 X_{BMI} + \varepsilon \quad (2)$$

- The ΔR^2 test suggested that the loss in fit between Model 1 and Model 2 was trivial.
- The Training MSE values suggested that Model 1 should be preferred.

What happens when we do the comparison based on Test MSE instead of Training MSE?

Training vs. Test MSE

```r
set.seed(235711)

## Read in the data:
dDat <- readRDS("../data/diabetes.rds")

## Split data into training and testing sets:
ind  <- sample(1 : nrow(dDat))
dat0 <- dDat[ind[1 : 400], ]   # Training data
dat1 <- dDat[ind[401 : 442], ] # Testing data

## Fit the models:
outF <- lm(bp ~ age + bmi + ldl + hdl, data = dat0)
outR <- lm(bp ~ age + bmi, data = dat0)

## Compute training MSEs:
trainMseF <- MSE(y_pred = predict(outF), y_true = dat0$bp)
trainMseR <- MSE(y_pred = predict(outR), y_true = dat0$bp)
```

Training vs. Test MSE

```
## Compute testing MSEs:
testMseF <- MSE(y_pred = predict(outF, newdata = dat1),
                y_true = dat1$bp)
testMseR <- MSE(y_pred = predict(outR, newdata = dat1),
                y_true = dat1$bp)
```

Compare the two approaches:

	Full	Restricted
Train	147.72	148.44
Test	141.25	138.02

MSE Values

Cross Validation

To train a model that best predicts new data, we use *cross-validation* to choose the best model from a pool of candidates.

- Given a set $\mathcal{F} = \{f_1(\mathbf{X}), f_2(\mathbf{X}), \ldots, f_J(\mathbf{X})\}$ of J competing models, the simplest form of cross-validation entails the following:

  1. Split the sample into two, disjoint sub-samples:
     - *Training* data
     - *Testing* data
  2. Estimate a candidate model, $f_j(\mathbf{X})$, on the training data.
  3. Check the predictive performance of $\hat{f}_j(\mathbf{X})$ on the testing data.
  4. Repeat Steps 2 and 3 for all candidate models in $\mathcal{F}$.
  5. Pick the $\hat{f}_j(\mathbf{X})$ that best predicts the testing data.

Estimating Prediction Error

The split-sample cross-validation scheme described above will underestimate prediction error.

- The same testing data are re-used to estimate the relative prediction error of the candidate models.
- If we need a good estimate of our final model's prediction error, we need to complicate the split-sample procedure slightly:

Estimating Prediction Error

The split-sample cross-validation scheme described above will underestimate prediction error.

- The same testing data are re-used to estimate the relative prediction error of the candidate models.
- If we need a good estimate of our final model's prediction error, we need to complicate the split-sample procedure slightly:
  1. Split the sample into *three* disjoint subsets:
     - *Training* data, *validation* data, and *testing* data
  2. Use the training and validation data to choose the best model via the cross-validation procedure described earlier.
  3. Fit the chosen model to the combined training and validation data.
  4. Use the Step 3 model to generate predictions from the testing data.
  5. Use the test-set predictions to estimate prediction error.

Estimating Prediction Error

```r
set.seed(235711)

## Split data into training, testing, and validation sets:
ind  <- sample(1 : nrow(dDat))
dat0 <- dDat[ind[1 : 350], ]    # Training data
dat1 <- dDat[ind[351 : 400], ] # Validation data
dat2 <- dDat[ind[401 : 442], ] # Testing data

## Fit models to training data:
outF <- lm(bp ~ age + bmi + ldl + hdl, data = dat0)
outR <- lm(bp ~ age + bmi, data = dat0)
```

Estimating Prediction Error

```
## Compute validation MSEs:
validMseF <- MSE(y_pred = predict(outF, newdata = dat1),
                 y_true = dat1$bp)
validMseR <- MSE(y_pred = predict(outR, newdata = dat1),
                 y_true = dat1$bp)

## Use validation MSEs to choose the best model:
validMseF

## [1] 163.6176

validMseR

## [1] 166.3149
```

Estimating Prediction Error

```
## Pool training and validation data:
dat01 <- rbind(dat0, dat1)
```

```
## Estimate the chosen model:
outC <- lm(bp ~ age + bmi + ldl + hdl, data = dat01)
```

```
## Estimate prediction error using the testing data:
predErr <- MSE(y_pred = predict(outC, newdata = dat2),
               y_true = dat2$bp)

## This value can be reported as our best estimate of the
## prediction error:
predErr

## [1] 141.2545
```

Different Flavors of Cross-Validation

In practice, the split-sample cross-validation procedure describe above can be highly variable.

- The solution is highly sensitive to the way the sample is split because each model is only training once.

Split-sample cross-validation can also be wasteful.

- We don't need to set aside an entire chunk of data for validation.

In most cases, we will want to employ a slightly more complex flavor of cross-validation:

- *K-fold cross-validation*

K-Fold Cross-Validation

1. If you need to estimate prediction error, set aside a testing set.
2. Partition the (remaining) data into K disjoint subsets $C_k = C_1, C_2, \ldots, C_K$.
3. Conduct K training replications.
   - For each training replication, collapse $K - 1$ partitions into a set of training data.
   - Compute the validation MSE for the kth partition, MSE_k, by using subset C_k as the validation data for the kth fitted model.
4. Compute the overall K-fold cross-validation error as:

$$CVE = \sum_{k=1}^{K} \frac{N_k}{N} MSE_k,$$

K-Fold Cross-Validation

```
getCve <- function(model, data, K, part) {
    ## Loop over K repetitions:
    mse <- c()
    for(k in 1 : K) {
        ## Partition data:
        train <- data[part != k, ]
        valid <- data[part == k, ]

        ## Fit model, generate predictions, and save MSE:
        fit    <- lm(model, data = train)
        pred   <- predict(fit, newdata = valid)
        mse[k] <- MSE(y_pred = pred,
                      y_true = valid[ , dvName(fit)])
    }
    ## Return the CVE:
    mean(mse)
}
```

K-Fold Cross-Validation

```
## Do K-Fold Cross-Validation with lm():
cv.lm <- function(data, models, K) {
    ## Create a partition vector:
    part <- rep(1 : K, nrow(data) / K)

    ## Permute the partition vector:
    part <- sample(part)

    ## Apply over candidate models:
    sapply(models, getCve, data = data, K = K, part = part)
}
```

K-Fold Cross-Validation

So how does *K*-fold cross-validation perform in our *BP* prediction task?

```
cvOut <- cv.lm(data   = dDat,
               models = c("bp ~ age + bmi + ldl + hdl",
                          "bp ~ age + bmi"),
               K      = 10)
cvOut

## bp ~ age + bmi + ldl + hdl                bp ~ age + bmi
##                   150.5036                      148.8830
```

Missing Data Imputation

Imputation is Just Prediction

In Lecture 3, you heard a bit about missing data imputation.

- Multiple imputation is one of the best ways to treat missing data.

Imputation is nothing more than a type of prediction.

1. Train a model on the observed parts of the data, Y_{obs}.
   - Train the imputation model.
2. Predict the missing values, Y_{mis}.
   - Generate imputations.
3. Replace the missing values with these predictions.
   - Impute the missing data.

Imputation can be used to support either prediction or inference.

- Our goals will dictate what type of imputation we need to do.

Levels of Uncertainty Modeling

Van Buuren (2012) provides a very useful classification of different imputation methods:

1. Simple Prediction
   - The missing data are naively filled with predicted values from some regression equation.
   - All uncertainty is ignored.
2. Prediction + Noise
   - A random residual error is added to each predicted value to create the imputations.
   - Only uncertainty in the predicted values is modeled.
   - The imputation model itself is assumed to be correct and error-free.
3. Prediction + Noise + Model Error
   - Uncertainty in the imputation model itself is also modeled.
   - Only way to get fully proper imputations in the sense of Rubin (1987).

Do we really need to worry?

The arguments against single imputation can seem archaic and petty. Do we really need to worry about this stuff?

Do we really need to worry?

The arguments against single imputation can seem archaic and petty. Do we really need to worry about this stuff?

- YES!!! (At least if you care about inference)

The following are results from a simple Monte Carlo simulation:

	Complete Data	Conditional Mean	Stochastic	MI
cor(X, Y)	0.500	0.563	0.498	0.497
Type I Error	0.052	0.138	0.120	0.054

Mean Correlation Coefficients and Type I Error Rates

Do we really need to worry?

The arguments against single imputation can seem archaic and petty. Do we really need to worry about this stuff?

- YES!!! (At least if you care about inference)

The following are results from a simple Monte Carlo simulation:

	Complete Data	Conditional Mean	Stochastic	MI
cor(X, Y)	0.500	0.563	0.498	0.497
Type I Error	0.052	0.138	0.120	0.054

Mean Correlation Coefficients and Type I Error Rates

- Conditional mean substitution overestimates the correlation effect.
- Both single imputation methods inflate Type I error rates.
- MI provides unbiased point estimates and accurate Type I error rates.

Simulate Some Toy Data

```
nObs <- 1000 # Sample Size
pm   <- 0.3  # Proportion Missing

sigma <- matrix(c(1.0, 0.5, 0.0,
                  0.5, 1.0, 0.3,
                  0.0, 0.3, 1.0),
                ncol = 3)

dat0 <- as.data.frame(rmvnorm(nObs, c(0, 0, 0), sigma))
colnames(dat0) <- c("y", "x", "z")
```

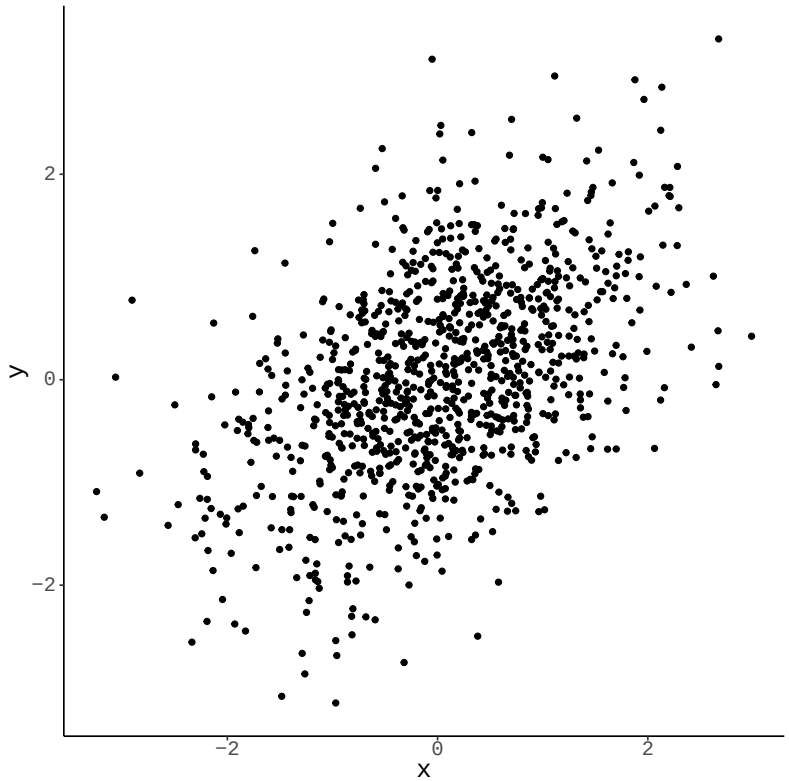
Simulate Some Toy Data

```
## Impose MAR Nonresponse:
dat1 <- dat0
rVec <- with(dat1, x < quantile(x, probs = pm))

dat1[rVec, "y"] <- NA

## Subset the data:
yMis <- dat1[rVec, ]
yObs <- dat1[!rVec, ]
```

Look at the data.

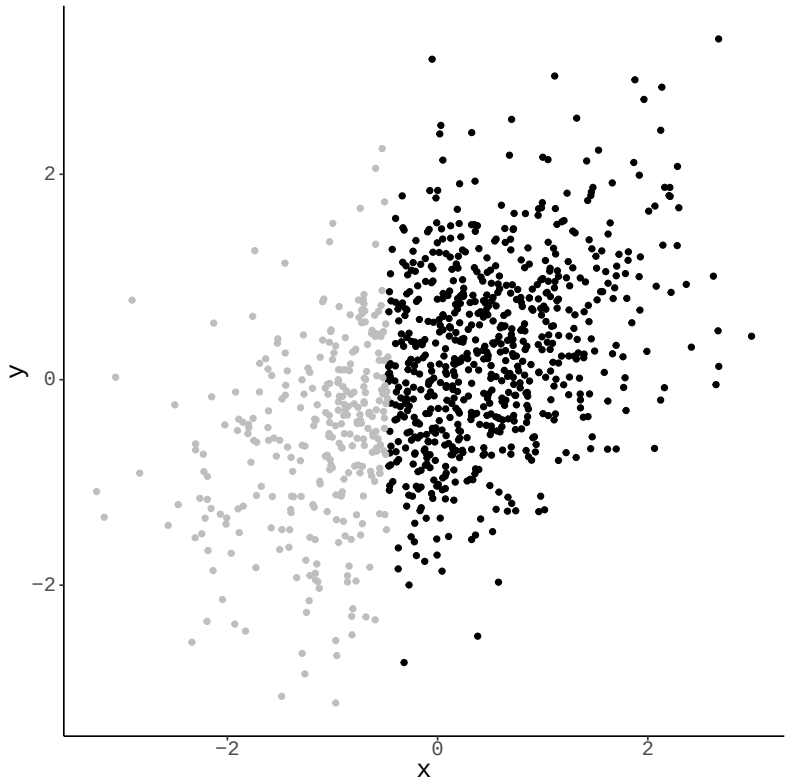
```
round(head(dat0, n = 5), 3)
```

```
##          y      x      z
## 1 -1.585 -0.942 -0.510
## 2  1.123 -0.161 -0.572
## 3  0.302  0.075 -1.980
## 4  0.325  0.850 -0.283
## 5  0.352 -0.400 -1.207
```



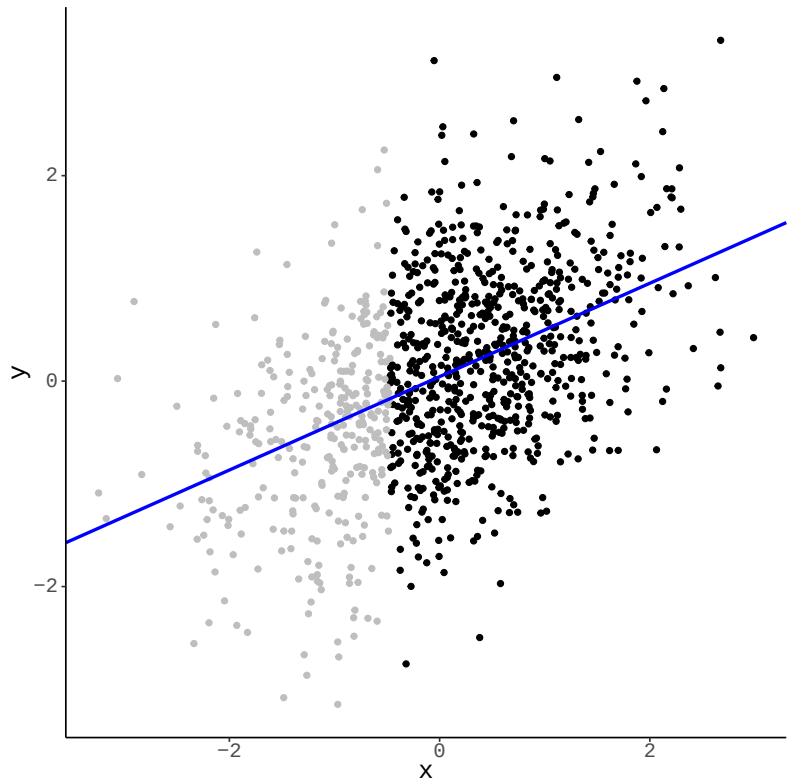
Look at the data.

```
round(head(dat1, n = 5), 3)

##         y      x      z
## 1      NA -0.942 -0.510
## 2 1.123 -0.161 -0.572
## 3 0.302  0.075 -1.980
## 4 0.325  0.850 -0.283
## 5 0.352 -0.400 -1.207
```



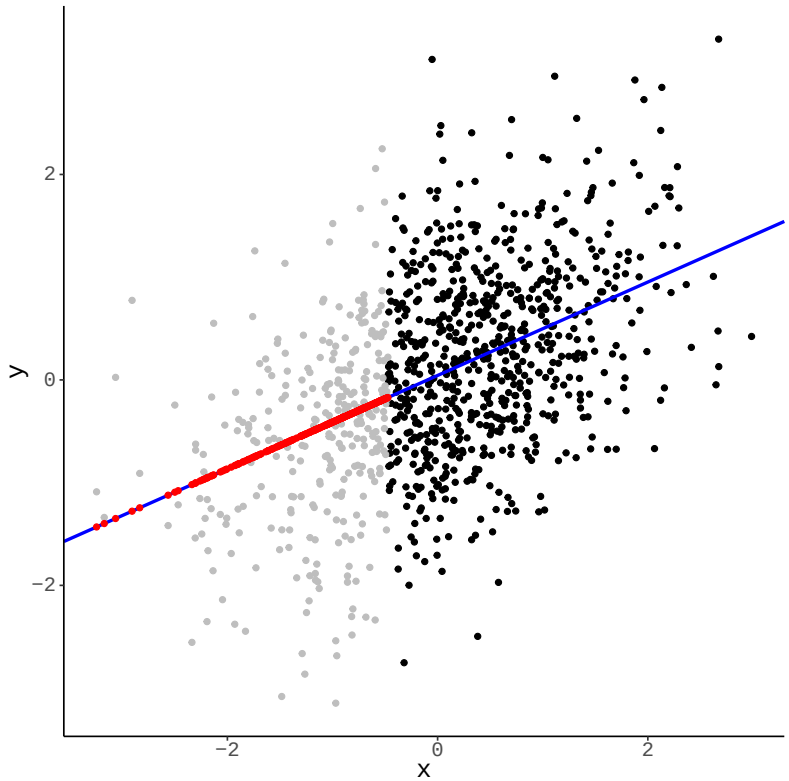
Expected Imputation Model Parameters

```
lsFit <- lm(y ~ x + z,
            data = yObs)

beta  <- coef(lsFit)
sigma <- summary(lsFit)$sigma

as.matrix(beta)

##                      [,1]
## (Intercept)  0.04575127
## x            0.49029969
## z           -0.15117793

sigma

## [1] 0.835231
```



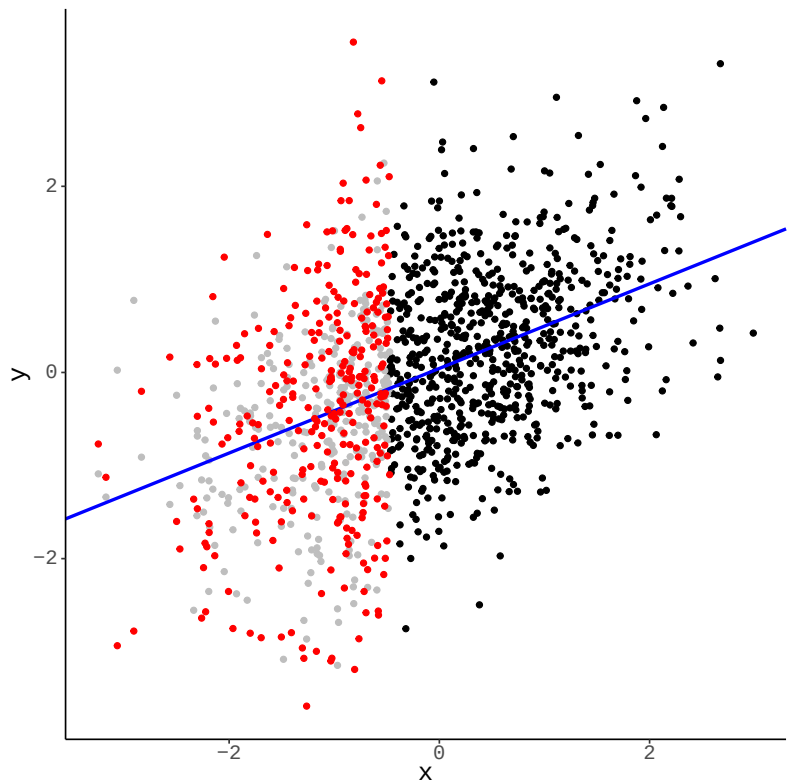
Conditional Mean Substitution

```
## Generate imputations:
imps <- beta[1] +
    beta[2] * yMis[ , "x"] +
    beta[3] * yMis[ , "z"]

## Fill missing cells in Y:
dat1[rVec, "y"] <- imps

round(head(dat1, n = 5), 3)

##          y      x      z
## 1 -0.339 -0.942 -0.510
## 2  1.123 -0.161 -0.572
## 3  0.302  0.075 -1.980
## 4  0.325  0.850 -0.283
## 5  0.352 -0.400 -1.207
```



Stochastic Regression Imputation

```
## Generate imputations:
imps <- imps +
    rnorm(nrow(yMis), 0, sigma)

## Fill missing cells in Y:
dat1[rVec, "y"] <- imps

round(head(dat1, n = 5), 3)

##       y      x      z
## 1 0.714 -0.942 -0.510
## 2 1.123 -0.161 -0.572
## 3 0.302  0.075 -1.980
## 4 0.325  0.850 -0.283
## 5 0.352 -0.400 -1.207
```



Flavors of MI

MI simply repeats a single regression imputation M times.

- The specifics of the underlying regression imputation are important.

Flavors of MI

MI simply repeats a single regression imputation M times.

- The specifics of the underlying regression imputation are important.

Simply repeating the stochastic regression imputation procedure described above won't suffice.

- Still produces too many Type I errors

	Complete Data	PN-Type	PNE-Type
cor(X, Y)	0.499	0.499	0.498
Type I Error	0.040	0.066	0.046

Mean Correlation Coefficients and Type I Error Rates

- Type I error rates for PN-Type MI are much better than they were for single stochastic regression imputation, but they're still too high.

Proper MI

The problems on the previous slide arise from using the same regression coefficients to create each of the M imputations.

- Implies that you're using the "correct" coefficients.
- This assumption is plainly ridiculous.
  - Any estimated regression line is only a guess of the true regression line.
  - Hence the standard errors of the coefficients.

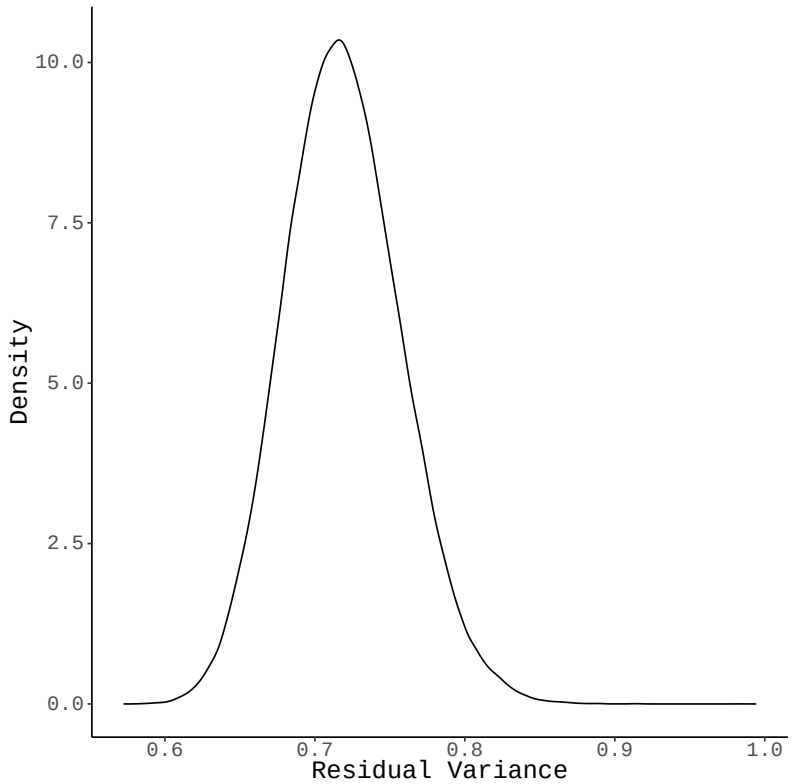
Proper MI

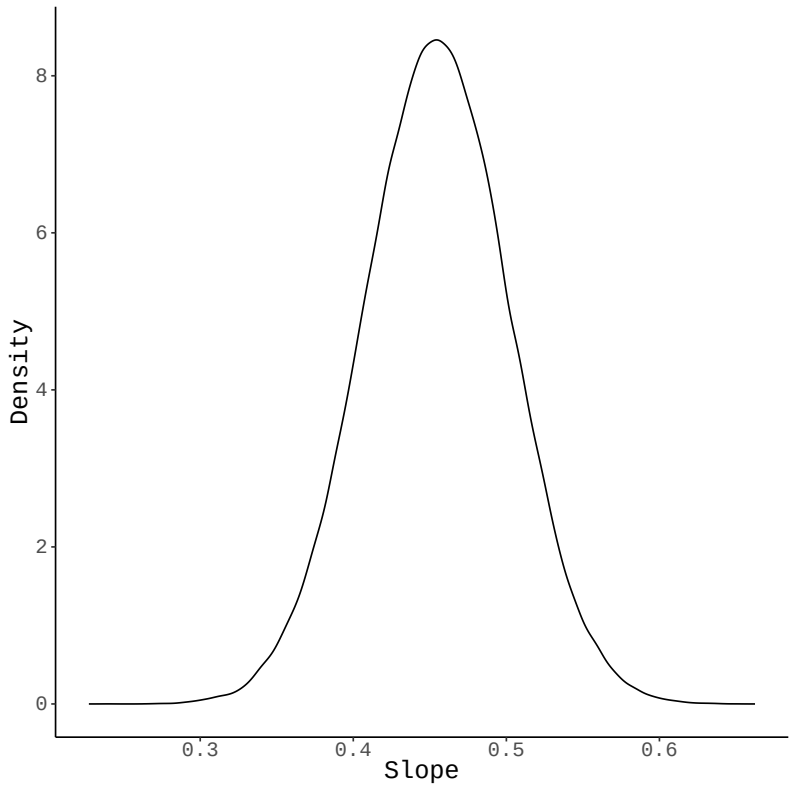
The problems on the previous slide arise from using the same regression coefficients to create each of the *M* imputations.

- Implies that you're using the "correct" coefficients.
- This assumption is plainly ridiculous.
  - Any estimated regression line is only a guess of the true regression line.
  - Hence the standard errors of the coefficients.
- Proper MI also models uncertainty in the regression coefficients used to create the imputations.
  - A different set of coefficients is randomly sampled (using Bayesian simulation) to create each of the *M* imputations.
  - The tricky part about implemented MI is deriving the distributions from which to sample these coefficients.

Visualizing MI

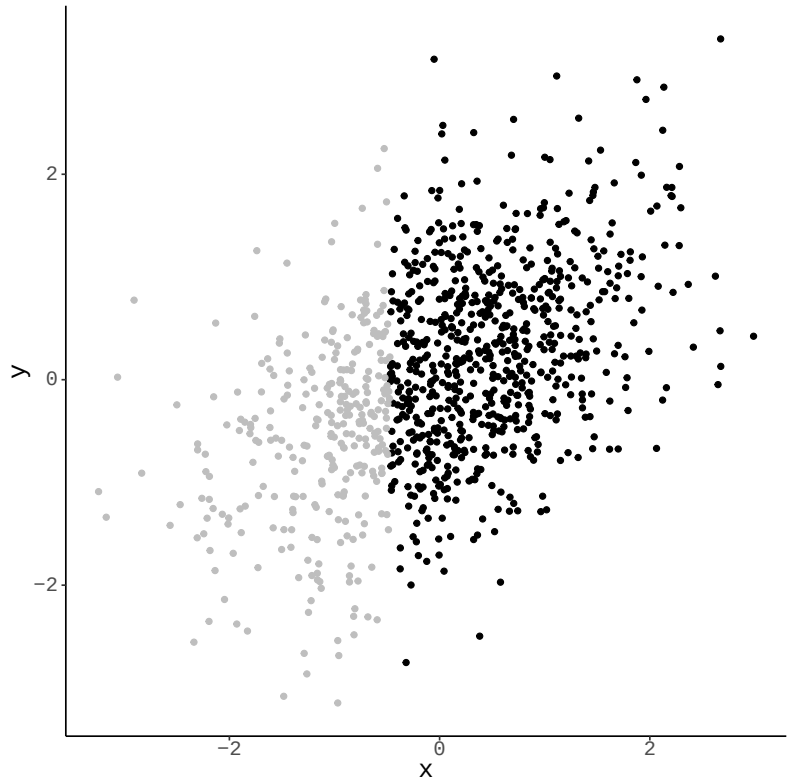
Use Bayesian simulation to estimate posterior distributions for the imputation model parameters:





Visualizing MI

Recall the incomplete data.

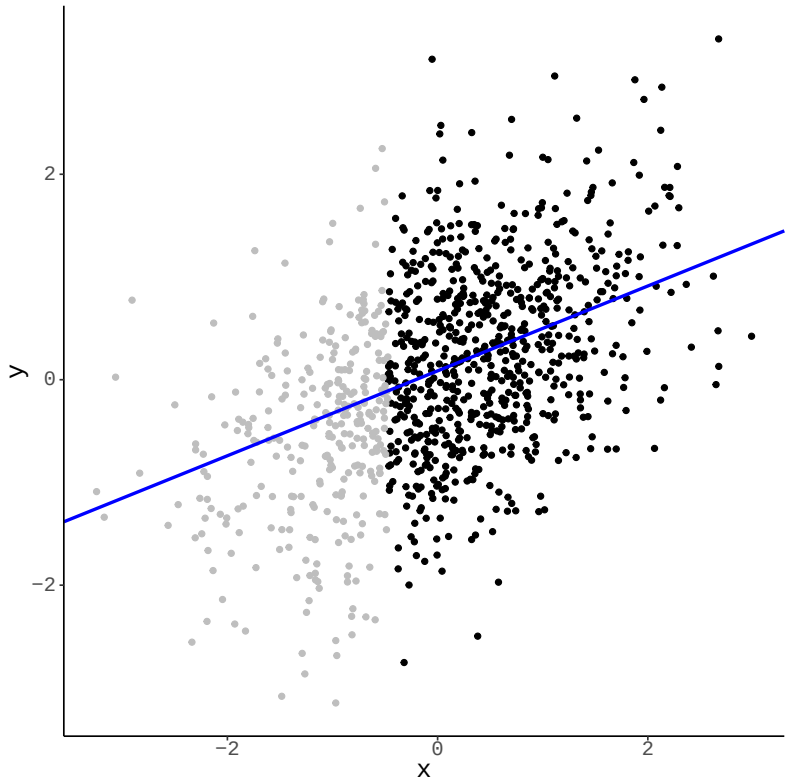


Visualizing MI

Sample values of β_0 and β_1:

- $\beta_0 = 0.086$
- $\beta_1 = 0.413$

Define the predicted best-fit line:
$\hat{Y}_{mis} = 0.086 + 0.413 X_{mis}$



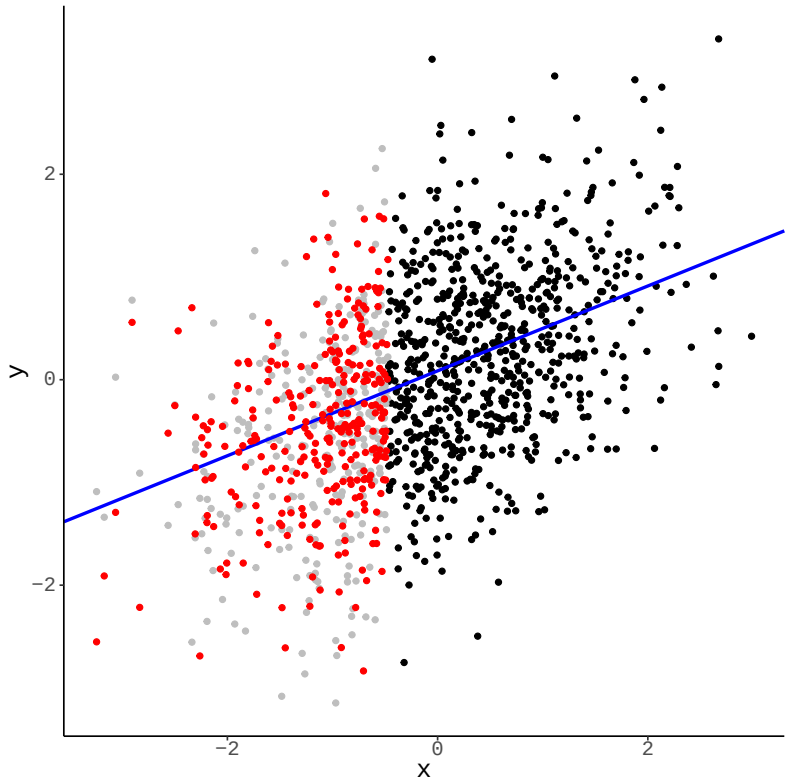
Visualizing MI

Sample a value of σ^2:

- $\sigma^2 = 0.724$

Generate imputations using the same procedure described in Single Stochastic Regression Imputation:

$$Y_{imp} = \hat{Y}_{mis} + \varepsilon$$
$$\varepsilon \sim \mathrm{N}(0, 0.724)$$

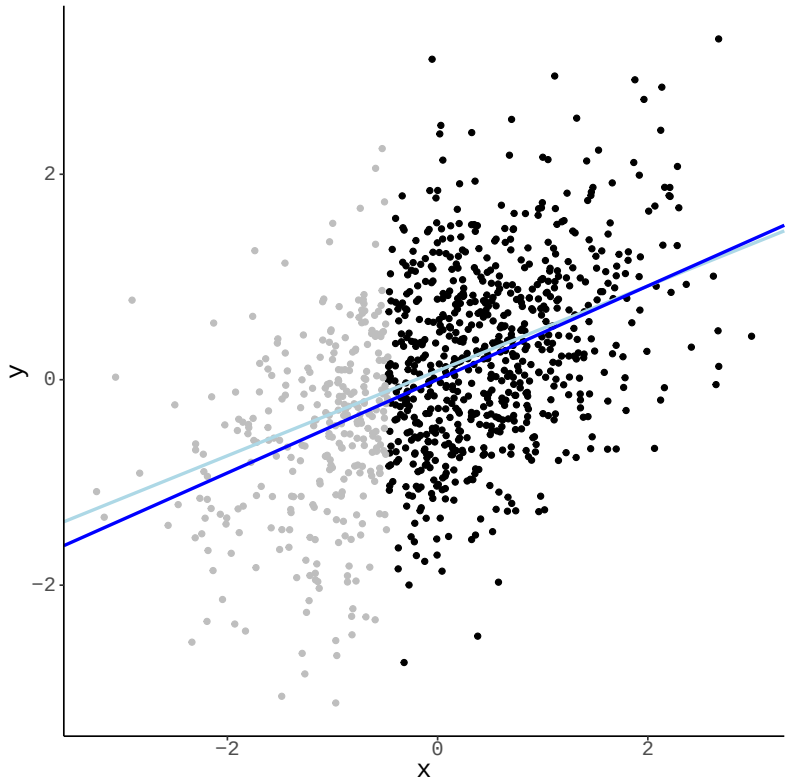


Visualizing MI

Sample values of β_0 and β_1:

- $\beta_0 = 0.003$
- $\beta_1 = 0.455$

Define the predicted best-fit line:
$\hat{Y}_{mis} = 0.003 + 0.455 X_{mis}$



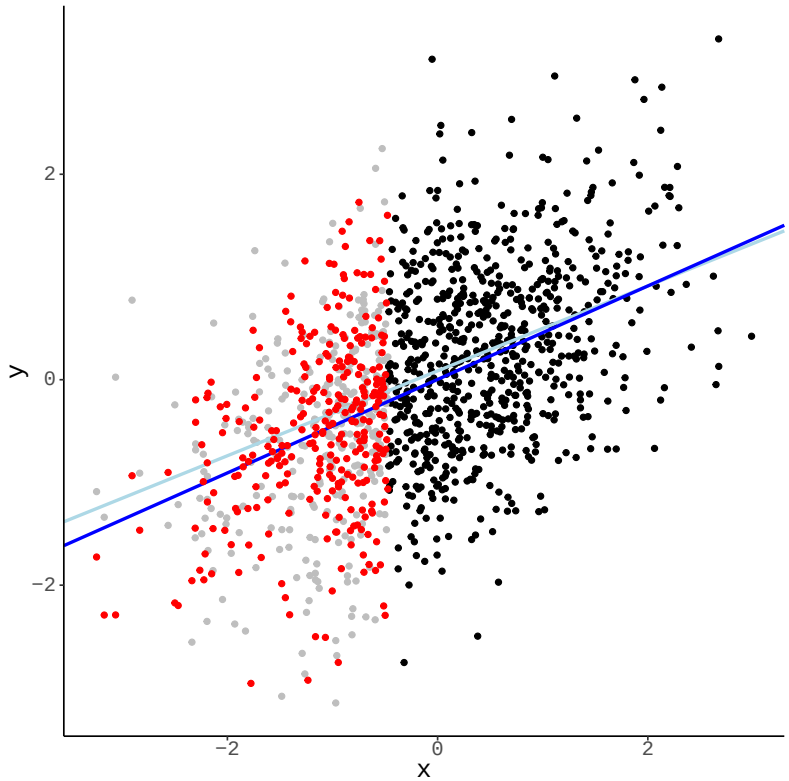
Visualizing MI

Sample a value of σ^2:

- $\sigma^2 = 0.788$

Generate imputations using the same procedure described in Single Stochastic Regression Imputation:

$$Y_{imp} = \hat{Y}_{mis} + \varepsilon$$
$$\varepsilon \sim \mathrm{N}(0, 0.788)$$

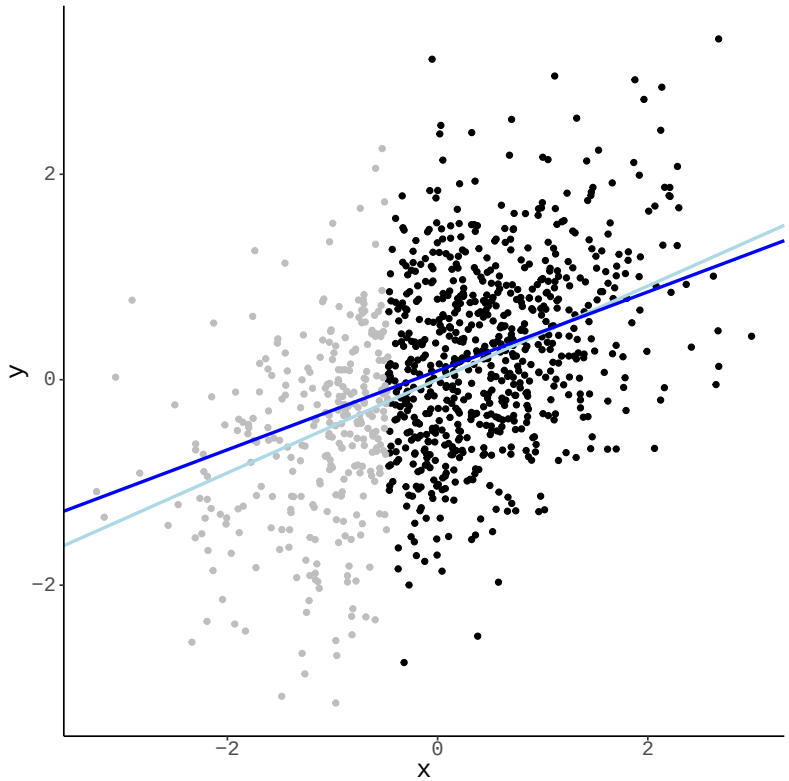


Visualizing MI

Sample values of β_0 and β_1:

- $\beta_0 = 0.086$
- $\beta_1 = 0.384$

Define the predicted best-fit line:
$\hat{Y}_{mis} = 0.086 + 0.384 X_{mis}$



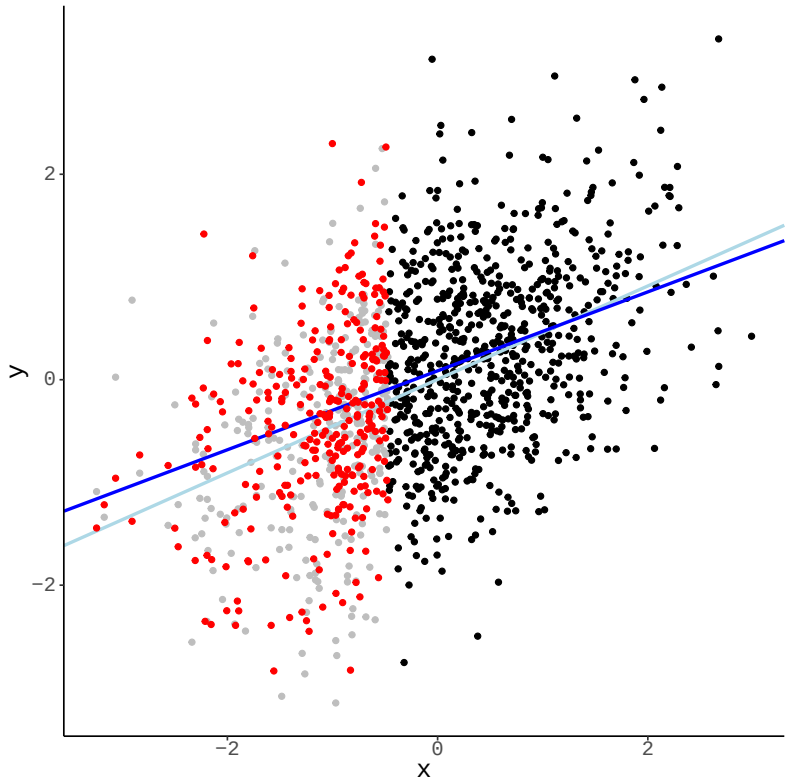
Visualizing MI

Sample a value of σ^2:

- $\sigma^2 = 0.716$

Generate imputations using the same procedure described in Single Stochastic Regression Imputation:

$$Y_{imp} = \hat{Y}_{mis} + \varepsilon$$
$$\varepsilon \sim \mathrm{N}(0, 0.716)$$



Doing Imputation in Practice

Each of the preceding approaches is available in the R package `mice` (Van Buuren & Groothuis-Oudshoorn, 2011).

```
## Conditional Mean Substitution:
miceOut1 <- mice(data   = missData,
                 m      = 1,
                 method = "norm.predict")
impDat1 <- complete(miceOut1)

## Stochastic Regression Imputation:
miceOut2 <- mice(data   = missData,
                 m      = 1,
                 method = "norm.nob")
impDat2 <- complete(miceOut2)

## Proper MI:
miceOut3 <- mice(data   = missData,
                 m      = 25,
                 method = "norm")
impList <- complete(miceOut3, "all")
```

Doing MI-Based Analysis

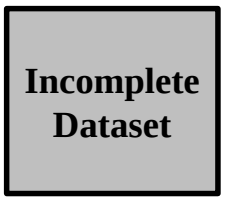
An MI-based data analysis consists of three phases:

1. The imputation phase
   - Replace missing values with M plausible estimates.
   - Produce M completed datasets.

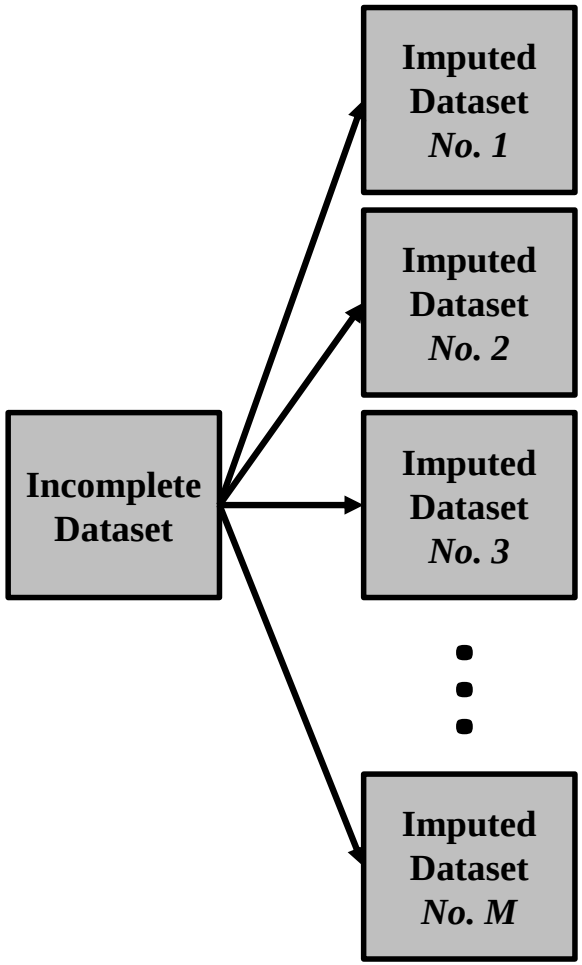
2. The analysis phase
   - Estimate M replicates of your analysis model.
   - Fit the same model to each of the M datasets from Step 1.

3. The pooling phase
   - Combine the M sets of parameter estimates and standard errors from Step 2 into a single set of MI estimates.
   - Use these pooled parameter estimates and standard errors for inference.

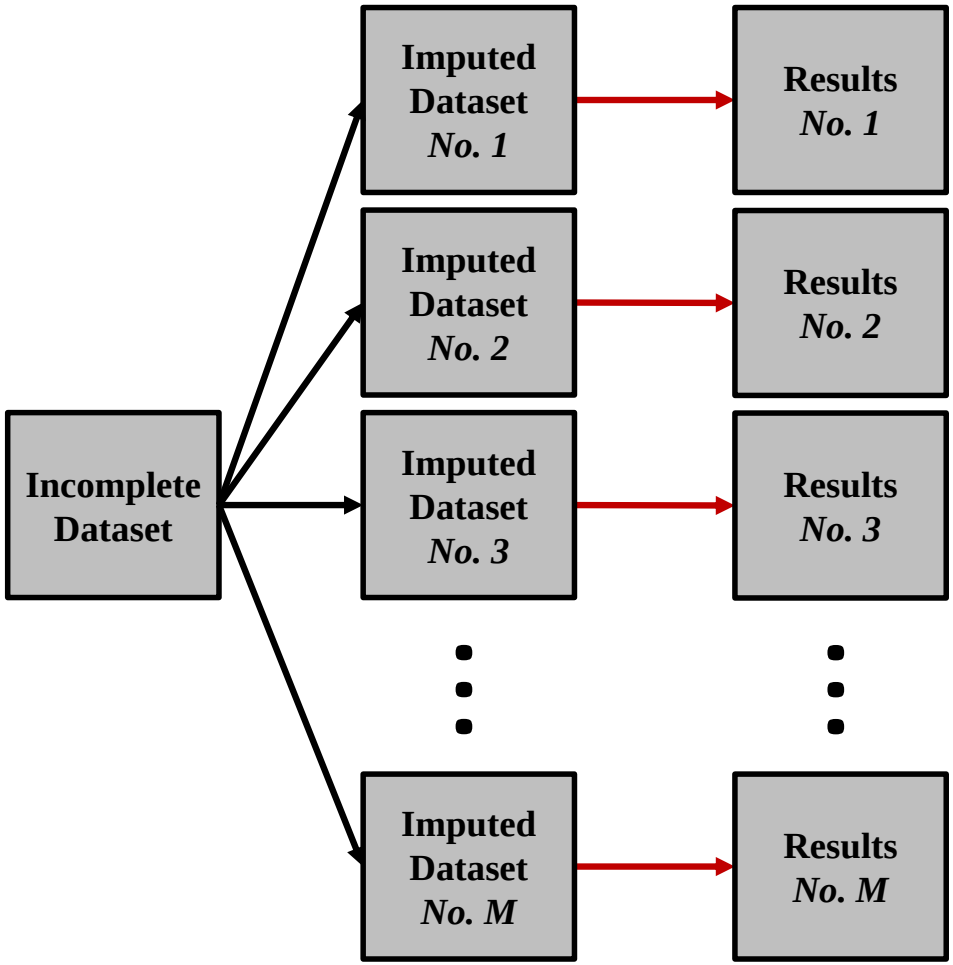
Schematic Representation of MI-Bases Analysis



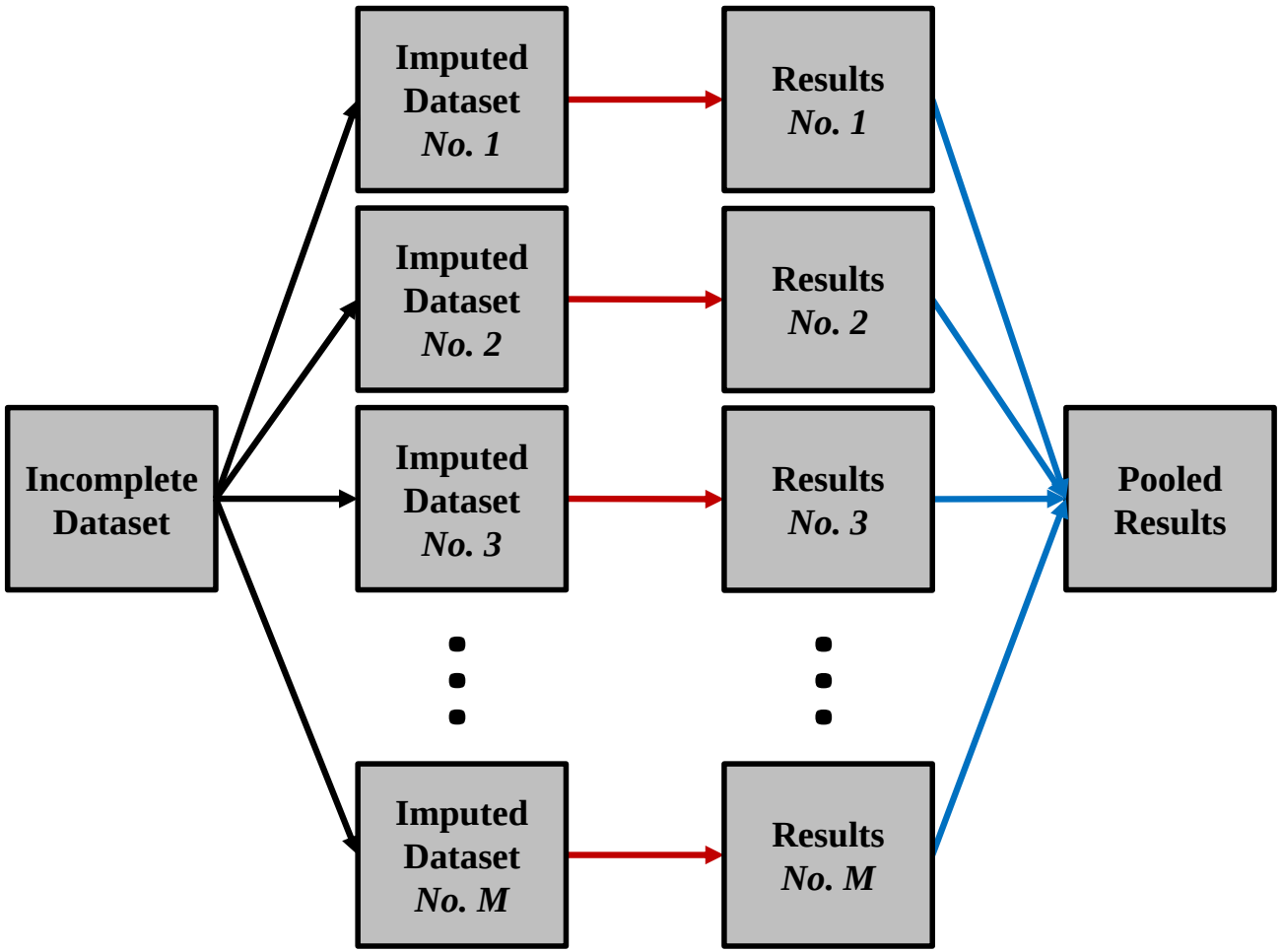
Schematic Representation of MI-Bases Analysis



Schematic Representation of MI-Bases Analysis



Schematic Representation of MI-Bases Analysis



Pooling MI Estimates

Rubin (1987) formulated a simple set of pooling rules for MI estimates.

- The MI point estimate of some interesting quantity, Q^*, is simply the mean of the M estimates, $\{\hat{Q}_m\}$:

$$Q^* = \frac{1}{M} \sum_{m=1}^{M} \hat{Q}_m$$

Pooling MI Estimates

The MI variability estimate, T, is a slightly more complex entity.

- A weighted sum of the *within-imputation* variance, W, and the *between-imputation* variance, B.

$$W = \frac{1}{M}\sum_{m=1}^{M} SE(\hat{Q}_m)^2$$

$$B = \frac{1}{M-1}\sum_{m=1}^{M}\left(\hat{Q}_m - Q^*\right)^2$$

$$T = W + \left(1 + M^{-1}\right) B$$

$$= W + B + \frac{B}{M}$$

Inference with MI Estimates

After computing Q^* and T, we combine them in the usual way to get test statistics and confidence intervals.

$$t = \frac{Q^* - Q_0}{\sqrt{T}}$$

$$CI = Q^* \pm t_{crit}\sqrt{T}$$

We must take care with our *df*, though.

$$df = (M-1)\left[1 + \frac{W}{(1+M^{-1})\,B}\right]^2$$

Fraction of Missing Information

In Lecture 4, we briefly discussed a very desirable measure of nonresponse: *fraction of missing information* (FMI).

$$FMI = \frac{r + \frac{2}{(df+3)}}{r+1} \approx \frac{(1+M^{-1})B}{(1+M^{-1})B + W} \rightarrow \frac{B}{B+W}$$

where

$$r = \frac{(1+M^{-1})B}{W}$$

The FMI gives us a sense of how much the missing data (and their treatment) have influenced our parameter estimates.

- We should report the FMI for an estimated parameter along with other ancillary statistics (e.g., t-tests, p-values, effect sizes, etc.).

Prediction/Cross-Validation with MI Data

When doing an MI-based analysis, we generally want to pool results as late as possible in the analytic process.

- This pattern also holds when doing prediction/cross-validation with MI data.
- When doing prediction, we pool the M sets of predictions.
  - We don't generate predictions using the pooled parameters.
- When doing cross-validation, we pool the M sets of MSE values.
  - We don't generate MSE values using pooled predictions or parameters.

Prediction/Cross-Validation with MI Data

To generate predictions from *M* multiply imputed datasets, we would:

1. Train the model on each of the *M* datasets separately.
2. Generate *M* sets of predictions from the *M* models trained above.
3. Average the *M* sets of predictions into a single vector of predicted values.

To run (K-fold) cross-validation with *M* multiply imputed datasets, we need to:

1. Split each of the *M* imputed datasets.
2. Run the cross-validation procedure separately on each of the *M* imputed datasets.
3. Pool the final *M* sets of MSE values.

Conclusion

- We can use a fitted regression model to generate predictions of new outcomes.
- We can get two different types of interval estimates around predictions: *confidence intervals* and *prediction intervals*.
- When building predictive models, we don't care that much about which variables are used.
  - We want to get the best possible predictions.
- When assessing predictive performance, we should consider *test-set* performance, not *training-set* performance.
- We can use (k-fold) cross-validation to tune our models and estimate prediction error.

Conclusion

- Missing data imputation is a very useful type of prediction.
- We can differentiate between three levels of uncertainty modeling in imputation: *P-*, *PN-*, and *PNE-type*.
  - To achieve correct inferences, we need to use full PNE-type multiple imputation.
- An MI-based analysis consists of three phases: *imputation*, *analysis*, and *pooling*.
- We should report the FMI for any parameter estimated in an MI-based analysis.

References

Rubin, D. B. (1987). *Multiple imputation for nonresponse in surveys* (Vol. 519). New York, NY: John Wiley & Sons.

Van Buuren, S. (2012). *Flexible imputation of missing data*. Boca Raton, FL: CRC Press. doi: 10.1201/b11826

Van Buuren, S., & Groothuis-Oudshoorn, K. (2011). mice: Multivariate imputation by chained equations in R. *Journal of Statistical Software*, *45*(3), 1–67.