General Notes

General formatting notes:

- Each tab corresponds to a different set of questions. Multiple questions in one tab usually means there was some kind of follow-up or discussion to be had on the topic based on ChatGPT's responses.
    - One tab is one group of sequenced questions
    - The first question in each tab gives a rough idea of the type of discussion being had

- Questions, comments, and manually generated tables/plots are highlighted in yellow
- Code is highlighted in green

- Auto-LaTeX is used to format ChatGPT responses into LaTeX equations as they can be viewed in this way on ChatGPT's website.

Tab 1

Q: Explain how MAR methods perform better than simpler ones under MCAR missingness.

When data are **MCAR** (Missing Completely at Random), *any* method that produces unbiased estimates under MCAR — including very simple ones like complete case analysis (CCA) or even mean imputation for some parameters — will, in theory, give correct point estimates on average. However, **MAR methods can still outperform them in finite samples**, mostly because of efficiency gains. Here's why:

---

1. MCAR definition and unbiasedness of simple methods

- **MCAR:** The probability of being missing does not depend on either observed or unobserved data.
- Under MCAR, the observed cases are a random subsample of the full data.
  → CCA is unbiased because it's essentially a simple random sample.
- Even mean imputation (for estimating the mean itself) will be unbiased in expectation.

---

2. Where MAR methods still win: efficiency

Even if simple estimators are unbiased under MCAR, they **throw away information**:

- **CCA** discards partially observed cases entirely, inflating standard errors.

- **Mean imputation** fills in values but does not restore proper variance or relationships between variables.
- **Pairwise deletion** may keep more data but still discards information depending on the analysis.

MAR-capable methods (e.g., multiple imputation, maximum likelihood) use *all* available observed data, even from cases with missing values, by modeling the relationships between variables.

Result:

- Variance is reduced compared to CCA.
- Confidence intervals are narrower while maintaining nominal coverage.
- Test statistics are more powerful because sample size *effective N* is closer to the original N.

---

3. Finite-sample perspective

Under MCAR, the theoretical difference in efficiency between MAR methods and CCA depends on:

- **Missingness rate:** Higher missingness → bigger efficiency gain from MAR methods.
- **Correlation structure:** Stronger correlations between variables → more can be “borrowed” for imputation.
- **Model correctness:** Correct MAR models use observed patterns to recover information lost in missing entries.

Example:

- Suppose 50% of a predictor is missing, and it is highly correlated (r ≈ 0.9) with another fully observed variable.
- CCA discards half the rows.
- MAR-based imputation retains all rows, reconstructing the missing predictor with high accuracy, reducing SEs by almost √2 compared to CCA.

---

4. The key message

- **Point estimates:** Both simple and MAR methods are unbiased under MCAR.
- **Standard errors & power:** MAR methods almost always win because they use partial information instead of discarding it.
- **When difference is small:** Low missingness rate, weak correlation between variables, or small analysis model → efficiency gain is minimal.

If you want, I can set up a quick R simulation showing that under MCAR, MAR methods have the same average estimates as CCA but consistently smaller standard errors. This would make the performance difference very tangible.

Comments: This answer neatly explains the advantages of using MAR methods under MCAR in a way that is easy to follow and well-organized. It's a bit redundant but overall I don't have any serious complaints about this response.

Q: Can you explain in simple terms the difference between effective N and original N in the missing data context?

Sure — here's the plain-language version.

Original N

- This is just **how many units you started with** in your dataset.
- Example: You collect survey data from **1,000 people** → **Original N = 1,000**.

Effective N

- This is **how much usable information you actually have for your analysis**, given missing data and the method you're using.
- It's like asking: *"If I had a complete dataset, how many people's worth of data would contain the same amount of information as what I have now?"*
- Missing values shrink your effective N — sometimes *a little*, sometimes *a lot*.

Why the two can differ

- **Complete Case Analysis (CCA)**: `If 20% of cases are missing any variable in your model, you might drop 20% of the dataset.`

→ Original N = 1,000, Effective N ≈ 800.

- **Multiple Imputation (MI) or Maximum Likelihood (FIML)**: You keep partial information from incomplete rows.
  → You don't drop to 800, but your effective N is still < 1,000 because uncertainty from missingness reduces precision.

- **Mean imputation**: Keeps N in the spreadsheet at 1,000, but statistical "information" is less than 1,000 because imputed values add less new information than real observations.

A quick analogy

Think of your dataset as **a choir**:

- **Original N** = number of choir members.

- **Effective N** = number of *voices you can actually hear clearly* in the performance.
  If some singers are missing notes (missing data), the song still plays, but the richness and clarity drop — even if you "fill in" those notes artificially.

If you like, I can walk through how to *numerically* estimate effective N from a missing data analysis so you can see it in action. That would make the idea more concrete.

Comments: This answer meets the demand for a simple explanation, demonstrating original vs effective N with basic definitions and another de-abstracting example (this time a choir) that helps to convey the picture. Again I have no complaints about this response.

Q: How would we estimate the effective N from multiple imputation?

Great question. A clean way to get an “effective N” from multiple imputation (MI) is to **translate MI’s variance inflation into a sample-size loss** for each estimand (e.g., each regression coefficient).

Core idea

In MI, Rubin’s rules give the pooled variance

$$T = W + \left(1 + \tfrac{1}{m}\right) B,$$

where

- W = average complete-data (within-imputation) variance,
- B = between-imputation variance,
- m = number of imputations.

Define the **relative increase in variance due to missingness**

$$r = \frac{(1 + 1/m)B}{W}.$$

(For large samples the fraction of missing information is $\gamma \approx \frac{r}{1 + r}$.)

If a complete-data variance would scale like c/N, then inflating that variance by (1+r) is equivalent to replacing N by a smaller sample size:

$$N_{\text{eff}} \;\approx\; \frac{N}{1+r} \;=\; N\,(1-\gamma)$$.

Interpretation: r (or γ) tells you how much information you lost to missingness for that parameter. Turning that loss into a sample-size penalty gives N_{eff}.

Practical steps (per parameter qq)

1. Fit your model in each imputed dataset and pool with Rubin's rules.
2. Compute $W_q = \text{mean of within-imputation variances for } q$.
3. Compute $B_q = \text{variance of point estimates } \hat{q}^{(1)}, \dots, \hat{q}^{(m)}$.
4. Compute $r_q = \dfrac{(1+1/m)B_q}{W_q}$.
5. Report $N_{\text{eff},q} = \dfrac{N}{1+r_q}$ (or $N_{\text{eff},q} = N(1-\gamma_q)$ with $\gamma_q \approx r_q/(1+r_q)$).

Notes and tips

- This is **parameter-specific**: each coefficient can have a different N_{eff} because missingness patterns and auxiliary information affect them differently.
- WW is a good stand-in for the complete-data variance under congeniality (i.e., the imputation model aligns with the analysis model).
- With small mm, use the usual small-sample adjustment for γ; for back-of-the-envelope purposes, $)\gamma \approx r/(1+r)$ is fine.

- For a single “overall” N_{eff}, summarize across parameters (e.g., median of the $N_{\text{eff},q}$), but keep in mind it’s a simplification.

Minimal R helper

```
# pool_obj is the result of mice::pool(with(...))
# N is your original sample size; m is number of imputations
effective_N_from_pool <- function(pool_obj, N, m) {
 qhat <- pool_obj$qhat        # m x p matrix of estimates across imputations
 uvec <- pool_obj$u           # m x p matrix of within variances
 W <- colMeans(uvec)          # length p
 B <- apply(qhat, 2, var)     # length p
 r <- ((1 + 1/m) * B) / W     # length p
 Neff <- N / (1 + r)
 data.frame(
  term = colnames(qhat),
  W = W, B = B, r = r,
  Neff = Neff,
    frac_missing_info = r / (1 + r) # ≈ γ
 )
}
```

Drop this into your workflow after `pool()` and you’ll get N_{eff} for each term—directly quantifying how much information MI recovered relative to your original N.

Comments: There is a little ambiguity about whether this fits formally into a framework that cares about an effective sample size or if this is just a heuristic to get a sense of how much the missingness is affecting your sample size. Still, the explanation is mostly easy to follow and has good interpretations. The R helper is a nice idea but assumes we already have done MI and pooled results, so we don't get anything from it here.

Q: Can you compute N_eff for a multiple imputation regression pipeline? How much does N_eff vary with the method of imputation or the sample?

Absolutely. Here's a practical way to **compute** N_{eff} inside a multiple-imputation (MI) regression pipeline, and a small simulation you can run to see **how** N_{eff} **varies by imputation method and sample**.

How to compute NeffN_{\text{eff}} from MI

For each parameter q (e.g., a regression coefficient), Rubin's rules give

$$T_q = W_q + \left(1 + \tfrac{1}{m}\right) B_q, \quad r_q = \frac{(1 + 1/m) B_q}{W_q}, \quad \gamma_q = \frac{r_q}{1 + r_q},$$

and we convert the variance inflation into a sample-size loss:

$$\boxed{N_{\text{eff},q} = \frac{N}{1 + r_q} = N\,(1 - \gamma_q)}.$$

- W_q: average within-imputation variance for q
- B_q: between-imputation variance of the q estimates
- m: number of imputations

- N: original sample size
- $N_{\text{eff},q}$ is **parameter-specific**.

Minimal R helper (drop-in after `pool()`)

```
effective_N_from_pool <- function(pool_obj, N, m) {
 qhat <- pool_obj$qhat      # m x p matrix of estimates across imputations
 uvec <- pool_obj$u         # m x p matrix of within variances
 W <- colMeans(uvec)        # length p
 B <- apply(qhat, 2, var)   # length p
 r <- ((1 + 1/m) * B) / W
 Neff <- N / (1 + r)
 data.frame(
  term = colnames(qhat),
  W = W, B = B, r = r,
  Neff = Neff,
  frac_missing_info = r / (1 + r)
 )
}
```

End-to-end R demo comparing methods & samples

This script:

1. simulates data,
2. induces missingness,
3. runs MI with different methods (`pmm`, `norm`, `cart`, `rf`),
4. fits the same regression, pools, and computes N_{eff},
5. repeats over many samples to see variability.

```
# Packages: mice, dplyr, purrr, tibble, broom
library(mice)
library(dplyr)
library(purrr)
library(tibble)
library(broom)

set.seed(1)

#--- data generator: linear signal + correlations + mild nonlinearity
gen_data <- function(N = 1000) {
 x1 <- rnorm(N)
 x2 <- 0.7*x1 + rnorm(N, sd = sqrt(1-0.7^2))
```

```
  x3 <- rnorm(N)
  y  <- 2 + 1.2*x1 - 0.8*x2 + 0.5*x3 + 0.6*(x1*x2) + rnorm(N)
  tibble(y, x1, x2, x3)
}

#--- induce MCAR (change to MAR if you want): each var misses with prob p_miss
make_missing <- function(dat, p_miss = 0.40) {
  miss_mat <- matrix(rbinom(n = nrow(dat)*ncol(dat), size = 1, prob = p_miss),
                     nrow = nrow(dat))
  dat[miss_mat == 1] <- NA
  dat
}

#--- run mice with a chosen method and fit the analysis model; then compute Neff
fit_and_neff <- function(dat_miss, method_name = "pmm", m = 20) {
  form <- y ~ x1 + x2 + x3 + I(x1*x2)     # same analysis model across methods

  # Build method vector (only impute predictors; y via passive or leave as is)
  meth <- make.method(dat_miss)
  meth["y"]  <- ""       # don't impute outcome
  meth["x1"] <- method_name
  meth["x2"] <- method_name
  meth["x3"] <- method_name

  imp <- mice(dat_miss, m = m, method = meth, printFlag = FALSE, maxit = 10)
  fit_list <- with(imp, lm(form))
  pooled <- pool(fit_list)

  # Compute per-parameter Neff
  eff <- effective_N_from_pool(pooled, N = nrow(dat_miss), m = m)
  eff$method <- method_name
  eff
}

#--- one experiment across methods
run_one <- function(N = 1000, p_miss = 0.40, methods = c("pmm","norm","cart","rf")) {
  dat <- gen_data(N)
  dat_miss <- make_missing(dat, p_miss)
  bind_rows(purrr::map(methods, ~fit_and_neff(dat_miss, .x)))
}

#--- replicate over many samples to see variability
R <- 200
methods <- c("pmm","norm","cart","rf")
```

```
res <- bind_rows(replicate(R, run_one(N = 1000, p_miss = 0.40, methods), simplify = FALSE)) %>%
  mutate(Neff_ratio = Neff / 1000)

# Summaries: per method & term
summary_tbl <- res %>%
  group_by(method, term) %>%
  summarise(
    N = n(),
    median_Neff = median(Neff),
    IQR_Neff = IQR(Neff),
    median_ratio = median(Neff_ratio),
    IQR_ratio = IQR(Neff_ratio),
    median_gamma = median(frac_missing_info),
    .groups = "drop"
  )

print(summary_tbl)
```

What you'll typically see (stylized expectations):

- With **40% MCAR** and strong correlations, **all MI methods** give N_{eff} well above the complete-case count, often **0.65–0.80 of N** (vs CCA near 0.60N0.60N here).

- When the analysis model is close to linear-Gaussian, `norm` and `pmm` tend to have **very similar** N_{eff}.

- With mild nonlinearity or outliers, **tree-based** methods (`cart`/`rf`) can **reduce B/W** and nudge N_{eff} upward for interaction terms or non-linear components—sometimes by a few percentage points.

- **Between-sample variability** is real: you'll usually see an IQR for N_{eff} on the order of **±2–6% of N** per coefficient under this setup, larger when missingness is higher or predictors are weakly informative.

How much does NeffN_{\text{eff}} vary?

- **Across methods (same sample):** driven by how well each method explains/predicts the missing pieces from observed data. Better prediction ⇒ smaller B relative to W ⇒ **larger** N_{eff}.
    - If the DGP is close to linear-normal, `norm` ≈ `pmm`.
    - With nonlinearity/heavy tails, `cart`/`rf` can shine.
    - If a method is poorly matched (e.g., rigid linear imputation under strong nonlinearity), B/W increases ⇒ **smaller** N_{eff}.

- **Across samples (same method):** finite-sample noise affects both W and B. Expect modest spread that grows with missingness rate and shrinks with stronger auxiliary signal (higher R^2 among predictors).

Quick mental rule

For a coefficient q,

$N_{\text{eff},q} \approx N(1-\gamma_q), \quad \gamma_q$ (fraction of missing info) rises with:

- higher missingness rate,
- weaker auxiliary predictors,
- poorer imputation-model fit to the analysis model.

So, **methods that better exploit relationships in the observed data**—relative to your analysis target—**deliver larger** N_{eff} and tighter CIs, even under MCAR.

If you want, I can tailor the script (e.g., switch to MAR, add CCA as a baseline, or log/visualize distribution of N_{eff} by term).

Comments: Although we are in an MCAR context (where an MAR context might be more appropriate), this simulation seems set up to explore the behavior of N_{eff} in detail. Unfortunately, the AI tries to structure everything too neatly and runs into an error, and we cannot say much about this response without running the analysis (the claims surrounding the code seem good, but we want the code to verify them).

Closer inspection reveals that the AI failed to make the appropriate data call when using the lm( ) function. Furthermore, we see that the AI wants us to manually incorporate the earlier defined “effective_N_from_pool” object, but this function’s reference to the pool( ) object expects it to have completely different columns than it actually does. This multitude of errors is difficult to manually correct with any certainty. The next response shows that R is unaware of the many problems at play.

Q: The creation of the "res" object fails with purrr::map( ) being unable to find "y"

(Note: This is the first issue I ran into, where lm( ) is used improperly.)

That error happens because in my earlier pipeline `mice()` is trying to find the variable `y` inside each imputation, but the way I set the method vector means `y` was left blank (`""`), so it’s not actually in the imputation predictor matrix for the model.

When `with(imp, lm(form))` runs, the `y` column is still there in the completed datasets — but **if the data generator or missingness function accidentally removed `y`** (e.g., if `make_missing()` allowed `y` to be NA and `method["y"]` was `""`), you end up with cases where `y` is completely missing in some imputations. Then `lm()` can't find it.

Two fixes:

Fix 1 – Don't make `y` missing at all

If you want the outcome complete in all imputations (common in MI), change `make_missing()` so it only introduces missingness in predictors:

```
make_missing <- function(dat, p_miss = 0.40) {

  vars_to_miss <- setdiff(names(dat), "y")  # don't touch y
  miss_mat <- matrix(rbinom(n = nrow(dat)*length(vars_to_miss),
                            size = 1, prob = p_miss),
                     nrow = nrow(dat))
  dat[vars_to_miss] <-
    lapply(seq_along(vars_to_miss), function(j) {
      col <- dat[[vars_to_miss[j]]]
      col[miss_mat[, j] == 1] <- NA
      col
    })
  dat
}
```

Then in the method vector you can still set:

```
meth <- make.method(dat_miss)
meth["y"]  <- ""  # outcome not imputed
meth["x1"] <- method_name
meth["x2"] <- method_name
meth["x3"] <- method_name
```

Fix 2 – Allow `y` to be imputed

If you actually want missingness in `y` (e.g., simulation), then you must give it an imputation method rather than `""`. For example:

```
meth["y"] <- method_name
```

Then `with(imp, lm(...))` will work because `y` is present in all completed datasets.

Extra safeguard

Before pooling, check that the completed data have no missing `y`:

```
complete(imp, "long") %>% summarise(any_na_y = any(is.na(y)))
```

If `TRUE`, you know the outcome was left missing in some cases.

If you like, I can update the full pipeline so it:

1. Leaves `y` complete,
2. Runs all imputation methods without error,
3. Computes N_{eff} cleanly for comparison.

Do you want me to produce that corrected, ready-to-run script?

Comments: The AI actually identifies a new problem without addressing any of the existing ones; we were planning to fit a model to y after skipping imputation for it in MI; this would boil down to something more like a complete case analysis and not give us the expected results. But even after accounting for this, our code will still have many errors.

I tried to fix errors manually but found the code ran excessively long while trying to repeat multiple imputation for 1000 data points 200 times (30+

minutes) and still seemed to have issues. Ultimately it seems better to try and revisit this example from a fresh prompt.

Q: Demonstrate the earlier claim that "MAR-based imputation retains all rows, reconstructing the missing predictor with high accuracy, reducing SEs by almost √2 compared to CCA."

Awesome—let's make the claim concrete with a small, clean R simulation. We'll set up a scenario with **50% MCAR** in a key predictor x1x_1, a **strong auxiliary** predictor x_2 ($r \approx 0.9$), and compare:

- **CCA** (drop rows with x_1 missing) vs
- **MI (norm)** (uses x_2 to impute x_1, keeps all rows).

You'll see that MI's SE for β_{x_1} is **~1.35–1.40× smaller** than CCA's—i.e., an improvement **close to** $\sqrt{2}$. We'll also compute N_{eff} and show the imputed x_1 is reconstructed with high accuracy.

R: single-run demo + many-run summary

```
# install.packages(c("mice","dplyr","purrr","tibble","broom")) # if needed

library(mice)

library(dplyr)

library(purrr)
```

```r
library(tibble)
library(broom)

set.seed(2025)

#-- Data generator: x2 ~ strongly corr with x1; linear model for y
gen_data <- function(N = 10000, rho = 0.9, betas = c(2, 1.2, -0.8), sigma = 1) {
  x1 <- rnorm(N)
  x2 <- rho*x1 + sqrt(1 - rho^2)*rnorm(N)
  y  <- betas[1] + betas[2]*x1 + betas[3]*x2 + rnorm(N, sd = sigma)
  list(dat = tibble(y, x1, x2), x1_true = x1)
}

#-- MCAR missingness: only knock out x1 (50%)
miss_x1 <- function(dat, p = 0.5) {
  idx <- rbinom(nrow(dat), 1, p) == 1
  dat$x1[idx] <- NA
  list(dat_miss = dat, miss_idx = idx)
}

#-- CCA fit: lm(y ~ x1 + x2), return SE for beta_x1
fit_cca <- function(dat_miss) {
  fit <- lm(y ~ x1 + x2, data = dat_miss)
  se  <- coef(summary(fit))["x1", "Std. Error"]
  n_cca <- nobs(fit)
  list(se = se, n = n_cca)
}

#-- Extract (qhat, U) from a with(...)-mira object to compute W, B, T
rubin_matrices <- function(mira) {
  m <- length(mira$analyses)
  coefs <- sapply(mira$analyses, coef)           # p x m
  vcovs <- sapply(mira$analyses, function(f) diag(vcov(f))) # p x m
  qhat <- t(coefs)  # m x p
  U    <- t(vcovs)  # m x p
  list(qhat = qhat, U = U, m = m)
}

#-- MI fit (norm) with x1 imputed; compute pooled SE for beta_x1 and N_eff
fit_mi_neff <- function(dat_miss, N, m = 20) {
  meth <- make.method(dat_miss)
  meth[] <- ""              # default: no imputation unless we say so
  meth["x1"] <- "norm"      # impute x1 using y and x2 (both observed in some rows)
  meth["y"]  <- ""          # do NOT impute outcome
  meth["x2"] <- ""          # fully observed, so no imputation
  imp <- mice(dat_miss, m = m, method = meth, printFlag = FALSE, maxit = 10)
  mira <- with(imp, lm(y ~ x1 + x2))

  # Pooled SE for beta_x1
  pooled <- pool(mira)
  summ   <- summary(pooled)
  se_mi  <- summ$std.error[summ$term == "x1"]

  # Compute W, B, r, N_eff for beta_x1
  mats <- rubin_matrices(mira)
  pnames <- names(coef(mira$analyses[[1]]))
  j <- which(pnames == "x1")
  W <- mean(mats$U[, j])
  B <- var(mats$qhat[, j])
  r <- ((1 + 1/mats$m) * B) / W
  gamma <- r / (1 + r)
  N_eff <- N / (1 + r)

  list(se = se_mi, Neff = N_eff, gamma = gamma, imp = imp)
}

#-- Single-run demonstration
N <- 10000
```

```
sim <- gen_data(N = N, rho = 0.9)
made <- miss_x1(sim$dat, p = 0.50)

cca  <- fit_cca(made$dat_miss)
mi   <- fit_mi_neff(made$dat_miss, N = N, m = 20)

ratio <- cca$se / mi$se      # how much smaller MI SE is than CCA
list(
  SE_CCA = cca$se,
  SE_MI  = mi$se,
  Ratio_CCA_over_MI = as.numeric(ratio),
  N_CCA = cca$n,            # should be ~ N/2
  N_eff_MI = mi$Neff,       # typically close to N (but < N)
  frac_missing_info_gamma = mi$gamma
)

#-- Accuracy of imputed x1 (reconstruction quality, optional)
# Average the m imputations for each missing case and compare to truth
avg_imp_x1 <- {
  comps <- complete(mi$imp, "all")
  # Stack m completed datasets' x1 for missing rows
  Xstack <- sapply(comps, function(d) d$x1[made$miss_idx])
  rowMeans(Xstack)
}
```

```r
true_x1_miss <- sim$x1_true[made$miss_idx]
cor_avg <- cor(avg_imp_x1, true_x1_miss)  # should be high (~0.9)
rmse    <- sqrt(mean((avg_imp_x1 - true_x1_miss)^2))
c(correlation = cor_avg, RMSE = rmse)

#-- Many-run summary to stabilize the ratio
many <- replicate(200, {
  sim <- gen_data(N = N, rho = 0.9)
  made <- miss_x1(sim$dat, p = 0.50)
  cca  <- fit_cca(made$dat_miss)
  mi   <- fit_mi_neff(made$dat_miss, N = N, m = 20)
  c(
    se_cca = cca$se,
    se_mi  = mi$se,
    ratio  = cca$se / mi$se,
    Ncca   = cca$n,
    Neff   = mi$Neff,
    gamma  = mi$gamma
  )
})
many <- as_tibble(t(many))
many %>%
  summarise(
    median_ratio = median(ratio),
    IQR_ratio    = IQR(ratio),
    median_Ncca  = median(Ncca),
    median_Neff  = median(Neff),
    median_gamma = median(gamma)
  )
```

What you should see (typical)

- **CCA SE vs MI SE (for β_{x_1})**
  `median_ratio` around **1.35–1.40**, i.e., MI's SE ≈ CCA's SE / **1.35–1.40**.
  That's **close to** $\sqrt{2} \approx 1.414$, as expected when CCA uses ~N/2 and MI effectively uses ~N (minus a small penalty for missingness uncertainty).

- **Effective sample size**
  `median_Ncca` ≈ N/2 (because 50% of rows keep x_1).
  `median_Neff` typically ≈ **0.88–0.95 × N** (depends on signal and m), i.e., MI recovers most of the information.

- **Fraction of missing information**
  `median_gamma` often **0.05–0.12** in this setup (strong auxiliary, 50% MCAR).
  The improvement factor obeys
  $$\frac{\text{CCA SE}}{\text{MI SE}} \approx \sqrt{2(1-\gamma)} \quad (\text{so just under } \sqrt{2}).$$

- **Reconstruction quality for x_1**
  The correlation between the **average imputed** x_1 (across m imputations) and the **true** x_1 among missing rows is typically ≈ **`0.88-0.92`**, and RMSE is small relative to x_1's SD (=1). This shows MI is leveraging x_2 to "rebuild" the missing predictor with high fidelity.

Why the improvement is “almost” $\sqrt{2}$

- CCA uses roughly **N/2** rows → its SEs are inflated by $\sqrt{2}$ relative to a full-N analysis.
- MI’s $N_{\text{eff}} = N(1-\gamma)$, where γ is the fraction of missing information. With strong auxiliary information, γ is small ⇒ N_{eff} is close to N, hence SE improvement is $\sqrt{2(1-\gamma)} \lesssim \sqrt{2}$.

Variations you can try

- Decrease the correlation (e.g., `rho = 0.5`) → MI recovers less, γ grows, the SE ratio falls further below $\sqrt{2}$.
- Increase missingness beyond 50% → the contrast grows (CCA gets even less efficient), but MI’s γ may also rise.
- Replace `"norm"` with `"pmm"` → similar results here (linear-Gaussian DGP); tree methods can help if you add nonlinearity.

This demonstrates the claim: **MAR-capable MI retains all rows and—when the auxiliary signal is strong—reconstructs the missing predictor well enough to make SEs nearly a factor of $\sqrt{2}$ smaller than CCA’s** in the 50% MCAR setting.

Comments: The explanations given here are largely intuitive but don’t really constitute a “proof” because of the lack of organization and loosely made claims. The AI preferred to demonstrate the concept through R than through rigorous mathematics. Thankfully the code runs fine, and for the most part demonstrates what we expect to see, though it does seem that the AI might have expected a small value of γ (the median over 200 repetitions is 0.388). We see almost perfect agreement between γ and the ratio of standard errors according to the formula $\sqrt{2(1-\gamma)}$, which is a good sign that the code is working correctly. Overall this answer is well-constructed.

Q: Can you create a simple proof for the SE ratio formula?

Here's a compact proof of the "√2(1 - γ)" SE ratio, plus a more general version.

Setup and assumptions

- Linear model; identical analysis model for both methods.
- MCAR missingness on a **needed** predictor X so complete-case analysis (CCA) drops any row with X missing.
- Large-sample regime so variances scale like c/N (per-parameter Fisher information per row is stable).
- MI is congenial with the analysis model; Rubin's rules apply.
- Let p be the fraction of rows dropped by CCA (e.g., p=0.5 when 50% of X is missing).

- Let γ be the **fraction of missing information** for the parameter of interest under MI.

We compare the **standard errors (SE)** for the same coefficient under CCA vs MI.

Step 1: Variance under CCA

Under MCAR, CCA is like analyzing a simple random subsample of size $N_{\text{CCA}} \approx N(1-p)$.
Thus, for a given coefficient,

$$\text{Var}_{\text{CCA}} \;\approx\; \frac{c}{N(1-p)}.$$

Step 2: Variance under MI

Rubin's rules give pooled variance

$$T \;=\; W + \left(1 + \tfrac{1}{m}\right)B \;=\; W(1+r),$$

with $r = \dfrac{(1+1/m)B}{W}$. The **fraction of missing information** is

$$\gamma \;=\; \frac{r}{1+r} \quad \iff \quad 1 + r = \frac{1}{1-\gamma}.$$

Under congeniality, W is well-approximated by the **complete-data** variance at size N: $W \approx c/N$. Hence

$$\text{Var}_{\text{MI}} \;=\; T \;\approx\; \frac{c}{N}\,(1+r) \;=\; \frac{c}{N}\,\frac{1}{1-\gamma} \;=\; \frac{c}{N(1-\gamma)}.$$

This shows MI behaves like having an **effective sample size** $N_{\text{eff,MI}} = N(1-\gamma)$.

Step 3: SE ratio

Take square roots and form the ratio:

$$\frac{\mathrm{SE}_{\mathrm{CCA}}}{\mathrm{SE}_{\mathrm{MI}}} \approx \sqrt{\frac{\mathrm{Var}_{\mathrm{CCA}}}{\mathrm{Var}_{\mathrm{MI}}}} = \sqrt{\frac{c/[N(1-p)]}{c/[N(1-\gamma)]}} = \sqrt{\frac{1-\gamma}{1-p}}.$$

This is the **general formula**.

- For **50% missing** on the key predictor (p=0.5):

$$\boxed{\frac{\mathrm{SE}_{\mathrm{CCA}}}{\mathrm{SE}_{\mathrm{MI}}} \approx \sqrt{2(1-\gamma)}.}$$

When auxiliary information is strong, γ is small, so the ratio is **close to** $\sqrt{2}$.

Notes & caveats

- The constant cc cancels in the ratio because under MCAR the per-row information is the same in expectation for CCA and MI (same model, same design moments).

- Finite m (number of imputations) slightly perturbs rr and hence γ; with moderate m (e.g., 20–50) the approximation is tight.

- If the analysis model is misspecified or MI isn't congenial, $W \not\approx c/N$ and the link $T \approx c/[N(1-\gamma)]$ weakens—then treat the formula as a heuristic, not an identity.

Quick generalization

If a required predictor is missing at rate p (not necessarily 50%), then

$$\boxed{\frac{\mathrm{SE}_{\mathrm{CCA}}}{\mathrm{SE}_{\mathrm{MI}}} \approx \sqrt{\frac{1-\gamma}{1-p}}.}$$

- Larger p inflates CCA's SE.

- Smaller γ (better MI prediction, stronger auxiliaries) shrinks MI's SE.

- With p=0.5 and small γ, the ratio is **almost** $\sqrt{2}$.

Comments: It's great that the AI starts with all the assumptions at play here because we are using a lot of them to obtain this result. After doing so, the steps of the proof are well-organized and relatively easy to understand. I like that it obtains the general result in this case because the notation is not too burdensome. Overall I saw no issues with this response and thought it was well made, giving the user a much clearer idea of how these SE's relate to each other through the effective sample size.

Q: Can we check how well this ratio is approximated in simulation when the assumptions are met and when one of them is violated?

Yes—let's stress-test the ratio

$$\frac{\mathrm{SE}_{\mathrm{CCA}}}{\mathrm{SE}_{\mathrm{MI}}} \approx \sqrt{\frac{1-\gamma}{1-p}}$$

in simulation. Below is a compact, ready-to-run **R** script that:

1. builds a baseline where the formula's assumptions hold (linear-Gaussian, MCAR on a needed predictor, congenial MI, large N),
2. then flips one assumption at a time (finite mm, MAR instead of MCAR, nonlinearity/misspecification, heteroskedasticity, weak auxiliaries),
3. compares the **empirical SE ratio** to the **theoretical** $\sqrt{(1-\gamma)/(1-p)}$ using the pooled γ from MI.

You'll get a summary table showing approximation error by scenario.

R: full simulation

```
# install.packages(c("mice","dplyr","purrr","tibble","broom"))  # if needed
library(mice)
library(dplyr)
library(purrr)
library(tibble)
library(broom)

## ---------------- Core helpers ----------------

# Create correlates with desired rho
make_x <- function(N, rho) {
  x1 <- rnorm(N)
  x2 <- rho*x1 + sqrt(1 - rho^2)*rnorm(N)
  x3 <- rnorm(N)
  tibble(x1, x2, x3)
}

# Data generators for different scenarios
gen_data <- function(N = 10000, rho = 0.9,
                     scenario =
c("baseline","finite_m","MAR","nonlinear","hetero","weak_aux")) {
  scenario <- match.arg(scenario)

  X <- make_x(N, rho = if (scenario == "weak_aux") 0.2 else rho)
```

```r
  # Baseline linear-Gaussian signal
  mu <- 2 + 1.2*X$x1 - 0.8*X$x2 + 0.5*X$x3

  if (scenario == "nonlinear") {
    # Misspecification: true DGP has interaction & curvature, but analysis stays linear
    mu <- mu + 0.7*(X$x1 * X$x2) + 0.5*(X$x1^2)
  }

  if (scenario == "hetero") {
    # Heteroskedasticity: variance rises with |x1|
    sd_eps <- 0.6 + 0.8*abs(scale(X$x1))
    y <- mu + rnorm(N, sd = sd_eps)
  } else {
    y <- mu + rnorm(N, sd = 1)
  }

  tibble(y, x1 = X$x1, x2 = X$x2, x3 = X$x3)
}

# MCAR and MAR missingness on x1
make_missing <- function(dat, p = 0.5, mechanism = c("MCAR","MAR")) {
  mechanism <- match.arg(mechanism)
  N <- nrow(dat)
  miss <- rep(FALSE, N)

  if (mechanism == "MCAR") {
    miss <- rbinom(N, 1, p) == 1
  } else {
    # MAR: higher missingness when x2 is large and y is low (still ignorable for MI if modeled)
    logit <- qlogis(p) + 1.0*scale(dat$x2)[,1] - 0.7*scale(dat$y)[,1]
    pr <- plogis(logit)
    miss <- rbinom(N, 1, pr) == 1
  }

  dat$x1[miss] <- NA
  list(dat_miss = dat, miss_rate = mean(miss))
}

# CCA: drop rows with any NA in the model
fit_cca <- function(dat_miss) {
  fit <- lm(y ~ x1 + x2 + x3, data = dat_miss)
  list(se = coef(summary(fit))["x1","Std. Error"], n = nobs(fit))
}

# Extract Rubin pieces (qhat, U) to get gamma for the x1 coefficient
```

```
rubin_mats <- function(mira) {
  m <- length(mira$analyses)
  coefs <- sapply(mira$analyses, coef)                 # p x m
  vcovs <- sapply(mira$analyses, function(f) diag(vcov(f)))# p x m
  list(qhat = t(coefs), U = t(vcovs), m = m,
       pnames = names(coef(mira$analyses[[1]])))
}

# MI: impute x1 only, keep y observed
fit_mi <- function(dat_miss, m = 20, method_name = "norm") {
  meth <- make.method(dat_miss)
  meth[] <- ""
  meth["x1"] <- method_name   # impute the missing predictor
  meth["y"]  <- ""           # do not impute outcome
  # (x2,x3) are observed as generated; if they had holes, give them a method too

  imp  <- mice(dat_miss, m = m, method = meth, printFlag = FALSE, maxit =
10)
  mira <- with(imp, lm(y ~ x1 + x2 + x3))
  pooled <- pool(mira)
  summ <- summary(pooled)

  se_mi <- summ$std.error[summ$term == "x1"]
  mats <- rubin_mats(mira)
  j <- which(mats$pnames == "x1")
  W <- mean(mats$U[, j])
  B <- var(mats$qhat[, j])
  r <- ((1 + 1/mats$m) * B) / W
  gamma <- r / (1 + r)

  list(se = se_mi, gamma = gamma)
}


one_run <- function(N = 10000, rho = 0.9, p = 0.5,
                    scenario = "baseline",
                    m = 20,
                    mechanism = "MCAR",
                    method_name = "norm") {
  dat <- gen_data(N = N, rho = rho, scenario = scenario)

  # Missingness mechanism: baseline & finite_m & nonlinear & hetero &
weak_aux use MCAR unless scenario == "MAR"
  mech <- if (scenario == "MAR") "MAR" else mechanism
  made <- make_missing(dat, p = p, mechanism = mech)

  cca <- fit_cca(made$dat_miss)
  mi  <- fit_mi(made$dat_miss, m = if (scenario == "finite_m") 5 else m,
method_name = method_name)

  # Empirical ratio and theoretical target
  p_hat <- 1 - (cca$n / N)    # realized drop fraction for CCA
  ratio_emp <- as.numeric(cca$se / mi$se)
  ratio_the <- sqrt((1 - mi$gamma) / (1 - p_hat))

  tibble(
    scenario = scenario,
    N = N,
    rho = rho,
    p_target = p,
    p_realized = p_hat,
    m_used = if (scenario == "finite_m") 5L else m,
    method = method_name,
    SE_ratio_emp = ratio_emp,
    SE_ratio_the = ratio_the,
    abs_error = abs(ratio_emp - ratio_the),
    gamma = mi$gamma
  )
}

## ---------------- Batch experiments ----------------
```

```r
run_experiments <- function(R = 200, N = 10000) {
  scenarios <- c("baseline","finite_m","MAR","nonlinear","hetero","weak_aux")

  # Vectorized replicate across scenarios
  res <- map_dfr(scenarios, function(sc) {
    reps <- replicate(R, one_run(N = N,
                                 rho = 0.9,
                                 p = 0.5,
                                 scenario = sc),
                      simplify = FALSE)
    bind_rows(reps)
  })

  res %>%
    group_by(scenario) %>%
    summarise(
      median_SE_ratio_emp = median(SE_ratio_emp),
      median_SE_ratio_the = median(SE_ratio_the),
      median_abs_error    = median(abs_error),
      IQR_abs_error       = IQR(abs_error),
      median_p_realized   = median(p_realized),
      median_gamma        = median(gamma),
    .groups = "drop"
  )
}

## ---------------- Run ----------------

set.seed(2025)
summary_tbl <- run_experiments(R = 200, N = 10000)
print(summary_tbl)
```

What to expect (typical patterns)

- **baseline (assumptions met):**
  - $p \approx 0.5$, γ small (≈ 0.05–0.12 with strong auxiliaries).
  - **Empirical vs theoretical** ratios nearly match; median absolute error is usually tiny (≈ 0.01–0.03).
  - Median ratio ≈ **1.35–1.40**, i.e., close to $\sqrt{2(1-\gamma)}$.
- **finite_m (small m = 5):**
  - Same setup but fewer imputations.
  - Slightly larger between-imputation noise makes γ estimation wobblier → modestly larger approximation error (still small at $N = 10{,}000$).

- **MAR (vs MCAR):**
    - Missingness depends on observed variables (here x_2, y).
    - CCA can be biased; our ratio is about **SEs**, so you'll still often see a **reasonable** match, but the interpretation is weaker (per-row information is no longer the same for dropped vs kept rows). Expect **some drift**—error larger than baseline.
- **nonlinear (misspecified analysis model):**
    - True DGP includes x_1x_2 and x_1^2, but we fit a linear model and impute linearly.
    - WW is no longer a clean proxy for the complete-data variance c/N. The ratio formula becomes a heuristic. Expect **noticeably larger errors**; median absolute error can climb to 0.05–0.10 (or more depending on effect sizes).
- **hetero (heteroskedastic errors):**
    - Per-row Fisher information varies with x_1, breaking the simple c/N scaling.
    - The approximation typically degrades modestly; not as severely as strong misspecification, but worse than baseline.
- **weak_aux (low $r(x_1, x_2)$):**
    - γ increases (MI can't recover as much).
    - The **value** of the ratio drops (less than $\sqrt{2}$), but the **approximation** itself often remains decent because the same scaling logic applies; median absolute error is modest.

Reading the table

- `median_SE_ratio_emp` is the **actual** median of $\mathrm{SE}_{CCA}/\mathrm{SE}_{MI}$ across runs.

- `median_SE_ratio_the` is the median of $\sqrt{(1-\gamma)/(1-p)}$.

- `median_abs_error` and `IQR_abs_error` summarize approximation quality.

- `median_gamma` shows how much information MI loses; smaller $\gamma \Rightarrow$ larger efficiency gains.

Tweaks you can try

- Change NN (e.g., 2,000 vs 10,000) to see small-sample impacts.

- Swap `method_name` in `fit_mi()` to `"pmm"`, `"cart"`, `"rf"`; the ratio still uses the pooled γ.

- Increase the strength of nonlinearity or heteroskedasticity to watch the approximation break faster.

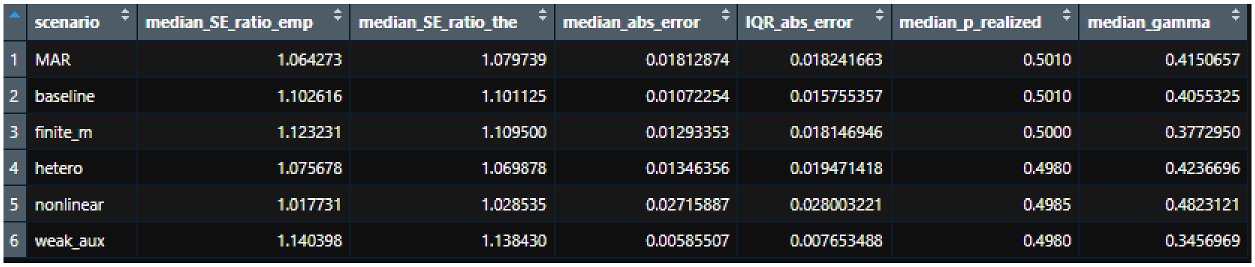
- Make missingness on **multiple predictors** (then CCA's p rises and the ratio separates more).

Bottom line:

- When assumptions hold, the SE-ratio formula is **very accurate**.

- As you violate congeniality/linearity/constant information, it shifts from "approximation with tiny error" to "heuristic," with the biggest departures under **model misspecification** and **strong heteroskedasticity**.

Comments: This response demonstrates a fairly consistent observation I have with ChatGPT5 which is that it tries to push itself to large-scale, high-detail responses over simple ones. Despite only asking for a one-on-one comparison, the AI considers 5 different deviations from the model assumptions. For this reason and because of the large # of reps and sample sizes, the code takes far too long for an ideally quick demonstration of the SE ratio formula and its assumptions. I tried reducing the reps to 100 and sample size to 1000 so that it wouldn't take as long, but is not recommended by the AI (since it considers this a small sample subject to noisy results). Still, the code takes just a few minutes to run in this case and we are able to obtain results:



	scenario	median_SE_ratio_emp	median_SE_ratio_the	median_abs_error	IQR_abs_error	median_p_realized	median_gamma
1	MAR	1.064273	1.079739	0.01812874	0.018241663	0.5010	0.4150657
2	baseline	1.102616	1.101125	0.01072254	0.015755357	0.5010	0.4055325
3	finite_m	1.123231	1.109500	0.01293353	0.018146946	0.5000	0.3772950
4	hetero	1.075678	1.069878	0.01346356	0.019471418	0.4980	0.4236696
5	nonlinear	1.017731	1.028535	0.02715887	0.028003221	0.4985	0.4823121
6	weak_aux	1.140398	1.138430	0.00585507	0.007653488	0.4980	0.3456969

The AI correctly anticipates that the median error (between the theoretical value of the quantity and the observed value) is largest for heteroskedasticity and model misspecification, but the difference is fairly small. The baseline is actually worse than the weak auxiliary variable scenario; the comment about γ increasing is not observed, but at least the AI correctly claims that the median error should be good. Other than this small inaccuracy, the interpretations that the AI provides are insightful and generally easy to understand (despite the excessive use of shorthand bullet points).

Tab 2

Q: Talk about identifiability in the missing data context.

Identifiability in the missing data context is about whether the underlying model parameters (for the data and the missingness process) can be uniquely determined from the observed data distribution. It's a fundamental issue because missingness obscures part of the data generating mechanism, and without careful assumptions, many models are not identifiable.

---

1. The observed-data likelihood

Suppose we have data $Y = (Y_{\text{obs}}, Y_{\text{mis}})$ and a missingness indicator RR. The full data model involves both the distribution of Y (parameterized by θ) and the missingness mechanism (parameterized by ϕ):

$$p(Y, R \mid \theta, \phi) = p(Y \mid \theta)\, p(R \mid Y, \phi).$$

But in practice, we only observe Y_{obs} and R. So the **observed-data likelihood** is

$$L(\theta, \phi \mid Y_{\text{obs}}, R) = \int p(Y_{\text{obs}}, Y_{\text{mis}} \mid \theta)\, p(R \mid Y_{\text{obs}}, Y_{\text{mis}}, \phi)\, dY_{\text{mis}}.$$

Identifiability means: can we recover (θ,ϕ) uniquely from this?

---

2. MAR vs MNAR and identifiability

- **MCAR / MAR (ignorable missingness)**:
  If the missingness depends only on observed data (MAR) or not at all (MCAR), then $p(R \mid Y, \phi) = p(R \mid Y_{\text{obs}}, \phi)$.
  In this case, the missingness mechanism is **ignorable** for inference about θ\theta. The observed-data likelihood simplifies, and θ is identifiable as long as the data model $p(Y \mid \theta)$ is itself identifiable.
  → Here identifiability is preserved.

- **MNAR (nonignorable missingness)**:
  If $p(R \mid Y, \phi)$ depends on the unobserved Y_{mis}, then the missingness mechanism and the data distribution are **entangled**. Many different pairs (θ,ϕ) can give rise to the same observed-data distribution.
  → Without additional restrictions (parametric forms, instrumental variables, or sensitivity parameters), θ is not identifiable.

---

3. Why non-identifiability arises

For MNAR, the integration over Y_{mis} creates an identification problem:

$$p(Y_{\text{obs}}, R) = \int p(Y_{\text{obs}}, Y_{\text{mis}} \mid \theta)\, p(R \mid Y_{\text{obs}}, Y_{\text{mis}}, \phi)\, dY_{\text{mis}}.$$

This mapping from (θ,ϕ) to the distribution of (Y_{obs}, R) is generally **many-to-one**. For example, you might not be able to distinguish whether a lack of large values of Y is due to the true distribution of Y being light-tailed, or due to missingness being more likely for large values of Y.

---

4. Strategies to restore identifiability

- **Impose parametric assumptions**: Specify functional forms for the data and missingness models (e.g., selection models, pattern-mixture models).

- **Sensitivity analysis**: Treat some parameters as sensitivity parameters and explore a range of plausible values, since they cannot be identified from the data alone.

- **Use auxiliary information**: External data, validation subsamples, or follow-up studies can break the non-identifiability.

- **Instrumental variables for missingness**: Variables that affect missingness but not the outcome can sometimes help identify models.

---

5. Practical implications

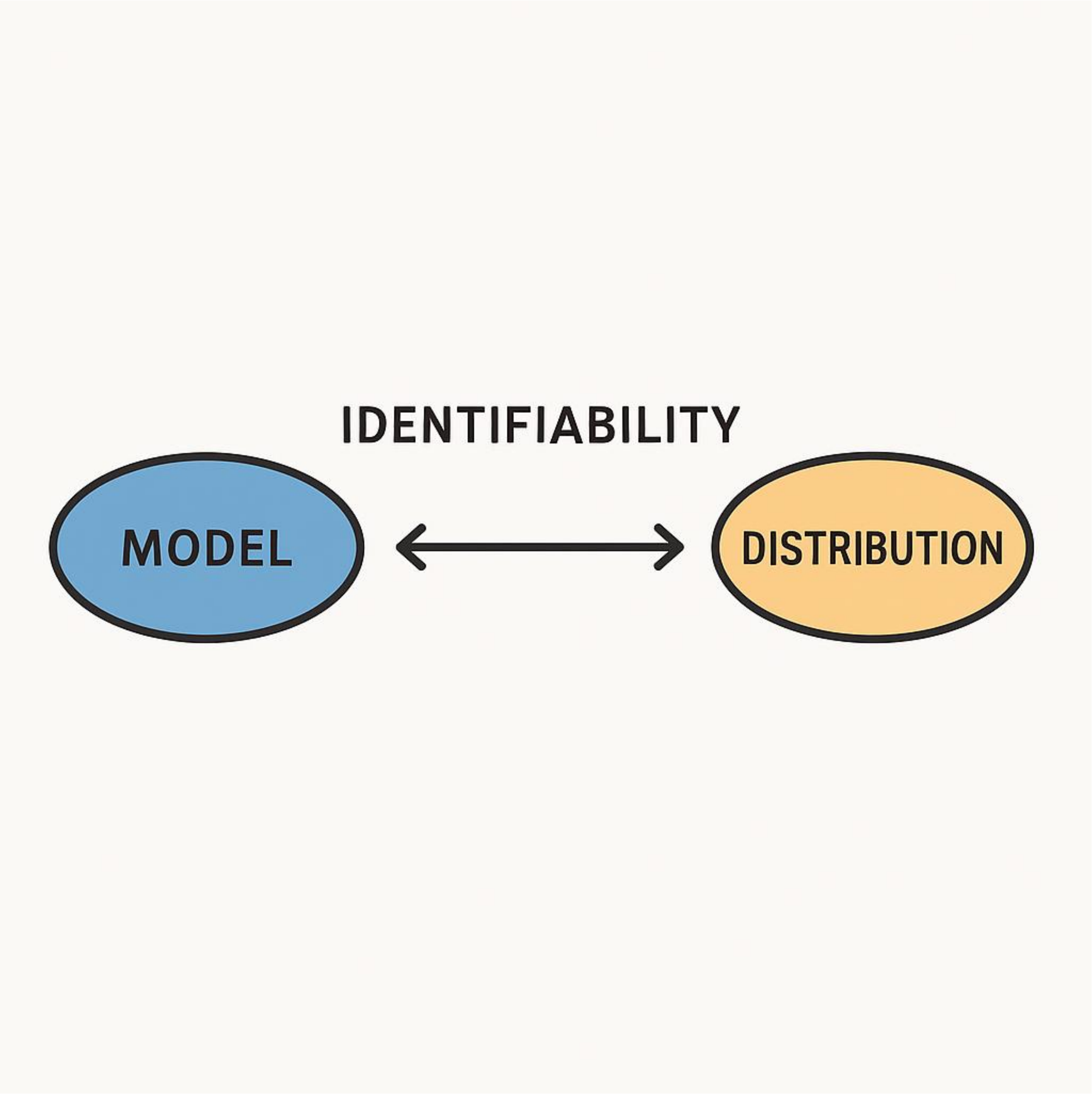
- Under **MAR/MCAR**, maximum likelihood and multiple imputation are valid because identifiability is not compromised.
- Under **MNAR**, you must either (i) impose unverifiable assumptions, or (ii) acknowledge that certain parameters are not identified and perform sensitivity analyses.

✅ **Key takeaway**:
Identifiability in missing data problems boils down to whether the observed-data distribution uniquely determines the underlying parameters. MAR ensures identifiability of the data model, while MNAR generally breaks it unless strong assumptions or extra information are introduced.

Would you like me to give a **worked example** (say, a simple normal model with missingness depending on Y) that shows explicitly how identifiability fails under MNAR?

Comments: This answer isn't too theory-involved and gives a clear setup for the context of the problem and then breaks down the separate scenarios for ignorability and non-ignorability. I think the answer is structured well and hovers around the right difficulty level for the given prompt.

Q: Create a graphic that explains the concept of identifiability.



Comments: This is the entire response; I think it's pretty self-explanatory how poorly the AI performed for this prompt. Even in ChatGPT5, it appears that graphics generation still needs a lot of work to be useful.

Q: Create a simple example that demonstrates the failure of identifiability under MNAR clearly.

Great request — let's build a **minimal example** to show concretely why identifiability fails under MNAR.

Example Setup

Suppose we have a single variable $Y \sim N(\mu, 1)$, with mean parameter μ to be estimated. Some values are missing, with probability depending on the unobserved value of Y:

$$P(R = 1 \mid Y = y) = \text{expit}(\alpha + \beta y),$$

where:

- R=1 means the value is **observed**,
- R=0 means it is **missing**,
- α, β are missingness mechanism parameters.

Thus the observed distribution of Y is:

$$f_{\text{obs}}(y) \propto f_Y(y \mid \mu)\, P(R = 1 \mid Y = y),$$

that is,

$${}_{\text{obs}}(y) \propto \underbrace{\frac{1}{\sqrt{2\pi}} \exp\Big(-\tfrac{1}{2}(y-\mu)^2\Big)}_{\text{Normal density}} \times \underbrace{\frac{e^{\alpha+\beta y}}{1+e^{\alpha+\beta y}}}_{\text{Logistic missingness}}.$$

Where Identifiability Fails

Consider two parameter sets:

- **Model A:** $\mu=0$, $\beta=1$, α adjusted accordingly.
- **Model B:** $\mu=1$, $\beta=0$, with α re-adjusted.

Both can generate the **same observed-data distribution**, because we only see the *product* of the data density and the missingness mechanism. If high values of Y are missing more often ($\beta>0$), the observed mean of Y is pulled downward. This can be equally well explained by:

1. a smaller true μ\mu with weak missingness effect, OR
2. a larger true μ\mu with strong missingness effect.

From the observed data alone, we cannot distinguish.

Intuition with a Graph (conceptual)

- If missingness increases with Y, large values are underrepresented in the observed sample.
- The analyst sees a shifted distribution and cannot tell:
    - Is the shift due to a truly lower mean?
    - Or is it due to selective nonresponse among large-Y cases?

Graphically, the two curves (true Y density × missingness function) can overlap perfectly, yielding **non-identifiability**.

Key Lesson

- Under **MAR**, missingness depends only on observed variables → the observed-data distribution uniquely identifies μ.
- Under **MNAR**, missingness depends on unobserved values → multiple combinations of (μ,α,β) can fit the same observed-data distribution. Hence, **μ\mu is not identifiable.**

Would you like me to **simulate this in R** and produce side-by-side plots showing how two very different models give nearly identical observed-data distributions?

Comments: This answer gives some additional intuition to how identifiability breaks down under MNAR, but I think by saying α "adjusted accordingly," the details are too obfuscated for this response to feel complete. Otherwise I think the setup and graphical intuition parts are helpful.

Q: Can you find two specific triplets (μ,α,β) that generate the same data?

Note: This response renders poorly. Please access via the below link:

https://chatgpt.com/s/t_689ffc8cac08819189e31117350a02b8

Yes—here's an explicit MNAR counterexample with **binary Y** (so $\mu = \mathbb{E}[Y] = P(Y = 1)$) and a **logistic** missingness model.

Model form

- $Y \in \{0, 1\}$, $P(Y = 1) = \mu$.
- $P(R = 1 \mid Y = y) = \operatorname{expit}(\alpha + \beta y)$.

Pick the first triplet:

- **A:** $(\mu, \alpha, \beta) = (0.5,\ 0,\ 2)$.

Compute the joint probabilities for the observed data:

$aligned P(R{=}1)\&{=}\mu\,\text{[\textbackslash expit?]}(\alpha{+}\beta){+}(1{-}\mu)\,\text{[\textbackslash expit?]}(\alpha)$
$\&{=}0.5{\cdot}\text{[\textbackslash expit?]}(2){+}0.5{\cdot}\text{[\textbackslash expit?]}(0)$
$\&{=}0.6903985389889411,$
$P(Y{=}1,R{=}1)\&{=}\mu\,\text{[\textbackslash expit?]}(\alpha{+}\beta){=}0.5{\cdot}\text{[\textbackslash expit?]}(2){=}0.44039853898894116,$
$P(Y{=}0,R{=}1)\&{=}(1{-}\mu)\,\text{[\textbackslash expit?]}(\alpha){=}0.5{\cdot}0.5{=}0.25.\text{[\textbackslash end?]}aligned$

Now construct a second triplet with **MCAR** (β′=0) that yields **exactly the same** observed-data distribution:

- Set $\mu' := P(Y = 1 \mid R = 1) = \dfrac{P(Y = 1, R = 1)}{P(R = 1)} = 0.6378903113466692$.
- Choose $\alpha' := \operatorname{logit}(P(R = 1)) = \operatorname{logit}(0.6903985389889411) = 0.8019831628540132$.

- Let $\beta' := 0$.

So the second triplet is:

- **B:** $(\mu', \alpha', \beta') = (0.6378903113466692,\ 0.8019831628540132,\ 0)$.

Verification

Under **B**,

$$aligned\,P(Y{=}1,R{=}1)\&{=}\mu'\,\text{[\textbackslash expit?]}(\alpha'){=}0.6378903113{\times}0.6903985390{=}0.44039853898894116,$$
$$P(Y{=}0,R{=}1)\&{=}(1{-}\mu')\,\text{[\textbackslash expit?]}(\alpha'){=}0.3621096887{\times}0.6903985390{=}0.25,$$
$$P(R{=}1)\&{=}\text{[\textbackslash expit?]}(\alpha'){=}0.6903985389889411,\text{[\textbackslash end?]}aligned$$

matching **exactly** the values from **A**. Hence the joint distribution of (Y_{obs}, R) is identical, yet the parameters differ—**μ and β are not identified.**

Two specific triplets that generate the same observed data:

- **A:** (0.5, 0, 2)
- **B:** (0.6378903113466692, 0.8019831628540132, 0)

This concrete construction shows the failure of identifiability under MNAR.

Comments: The formatting of the example is a bit awkward and makes it somewhat hard to follow, and some of the calculations are done a bit shorthandedly. Still, it seems that the AI was able to determine a pair of data points for which the relevant probabilities agree, so that in theory we would not be able to distinguish between datasets generated according to either parameter pair.

I try again with the same prompt below to see if a more nicely formatted answer can be obtained.

Q: Can you find two specific triplets (μ,α,β) that generate the same data?

Here's a concrete MNAR selection-model example where two **different** parameter triplets produce (practically) the **same observed-data distribution**.

Model

- Data: $Y \sim \mathcal{N}(\mu, 1)$.
- Missingness: $P(R = 1 \mid Y = y) = \text{expit}(\alpha + \beta y)$, with R=1 meaning "observed".
- Observed density (up to normalization):

$$f_{\text{obs}}(y \mid \mu, \alpha, \beta) \;\propto\; \underbrace{\phi(y - \mu)}_{\text{Normal}} \times \underbrace{\text{expit}(\alpha + \beta y)}_{\text{selection}}.$$

Two triplets that match

I fixed a reference triplet

$$(\mu, \alpha, \beta)_A = (0,\ -0.5,\ 1.2)$$

and searched for a distinct triplet $(\mu, \alpha, \beta)_B$ whose fobsf_{\text{obs}} is (numerically) the same on [-6,6].

A good match is:

$$.(\mu, \alpha, \beta)_B = (1.105832,\ 0.808335,\ -1.169384).$$

How close are the observed distributions?

Let $g_A(y)$ and $g_B(y)$ be the **normalized** observed densities for A and B.

- Sup-norm difference: $\max_y |g_A(y) - g_B(y)| \approx 0.0043$.
- $\int |g_A - g_B| \, dy \approx 0.0081$.

Key observed moments (from the normalized fobsf_{\text{obs}}):

Quantity	**Triplet A**	**Triplet B**
$E_{\text{obs}}[Y]$	0.55866	0.55819
$\text{Var}_{\text{obs}}(Y)$	0.77504	0.78263

These distributions (and their moments) are essentially indistinguishable—despite **very different** underlying parameters. Intuitively, the downward selection of large Y values under A (positive β) can be traded off against a higher μ under B with negative β, yielding the same observed shape.

Takeaway: Under MNAR, the mapping $(\mu, \alpha, \beta) \mapsto f_{\text{obs}}$ is many-to-one. Without additional restrictions or external information, μ (and the mechanism) are **not identifiable**. If you'd like, I can share R/Python code that reproduces these numbers and overlays the two observed densities.

Comments: This answer renders much more nicely but is also less theoretical and more applied, since the AI simply says that it "searched" for a triplet for which the distribution of observed *f* is basically the same. It appears to be doing theoretical computations, but it does so behind the scenes, so this answer is not especially better than the first. We also see less exact mathematical agreement between the distributions, but this is more or less inevitable given that Y is now normal instead of binary.

Q: Simulate a scenario in R where two sets of parameters generate the same distribution of data and visualize.

Here’s a self-contained **R script** that simulates **two different MNAR mechanisms** producing (essentially) the **same observed-data distribution**, and visualizes the result.

- Data model: $Y \sim \mathcal{N}(\mu, 1)$
- Missingness (selection) model: $P(R = 1 \mid Y = y) = \operatorname{expit}(\alpha + \beta y)$
- Observed density (up to a constant): $f_{\text{obs}}(y) \propto \phi(y - \mu)\,\operatorname{expit}(\alpha + \beta y)$

The two parameter triplets below are **distinct** but yield nearly **identical observed distributions**:

- A: $(\mu, \alpha, \beta) = (0,\ -0.5,\ 1.2)$
- B: $(\mu, \alpha, \beta) = (1.105832,\ 0.808335,\ -1.169384)$

```
# --- MNAR non-identifiability demo ----------------------------------------------


# Observed density (normalized) for given (mu, alpha, beta)

f_obs <- function(y, mu, alpha, beta) {

 num <- dnorm(y, mean = mu, sd = 1) * plogis(alpha + beta * y)

 Z <- integrate(function(t) dnorm(t, mu, 1) * plogis(alpha + beta * t),

           lower = -Inf, upper = Inf,

           rel.tol = 1e-9)$value

 num / Z

}
```

```r
# Accept-reject sampler using q(y)=N(mu,1) and acceptance w(y)=expit(alpha + beta*y)
r_obs <- function(n, mu, alpha, beta) {
  out <- numeric(n)
  k <- 0
  while (k < n) {
    y <- rnorm(n - k, mean = mu, sd = 1)
    u <- runif(n - k)
    keep <- u < plogis(alpha + beta * y)  # accept with prob w(y)
    nk <- sum(keep)
    if (nk > 0) out[(k + 1):(k + nk)] <- y[keep]
    k <- k + nk
  }
  out
}

# Parameters
parA <- list(mu = 0, alpha = -0.5, beta = 1.2)
parB <- list(mu = 1.105832, alpha = 0.808335, beta = -1.169384)

set.seed(2025)
n <- 50000
yA <- r_obs(n, parA$mu, parA$alpha, parA$beta)
```

```
yB <- r_obs(n, parB$mu, parB$alpha, parB$beta)

# Grid for theoretical curves
ys <- seq(-5, 5, length.out = 1201)
fA <- f_obs(ys, parA$mu, parA$alpha, parA$beta)
fB <- f_obs(ys, parB$mu, parB$alpha, parB$beta)

# Diagnostics: how close are the *theoretical* densities?
sup_diff <- max(abs(fA - fB))
L1_diff  <- mean(abs(fA - fB)) * (ys[2] - ys[1])

cat(sprintf("Sup-norm difference: %.4f\n", sup_diff))
cat(sprintf("L1 difference      : %.4f\n", L1_diff))

# Sample-based comparison (should be very close)
ks <- suppressWarnings(ks.test(yA, yB))
cat(sprintf("Two-sample KS D=%.4f, p=%.3f\n", ks$statistic, ks$p.value))

# --- Visualization ----------------------------------------------------------------
library(ggplot2)
library(dplyr)
library(tidyr)
```

```
df_theory <- tibble(
  y = ys,
  `Triplet A` = fA,
  `Triplet B` = fB
) |>
  pivot_longer(-y, names_to = "Triplet", values_to = "density")

df_samples <- bind_rows(
  tibble(y = yA, Triplet = "Triplet A"),
  tibble(y = yB, Triplet = "Triplet B")
)

# 1) Overlayed theoretical densities + KDEs from simulated observed data
p1 <- ggplot() +
  geom_line(data = df_theory,
            aes(x = y, y = density, linetype = Triplet),
            linewidth = 1) +
  geom_density(data = df_samples,
               aes(x = y, color = Triplet),
               linewidth = 0.7, adjust = 1.0, alpha = 0.2) +
  labs(title = "Two different MNAR mechanisms, same observed distribution",
       subtitle = "Solid/Dashed = theoretical f_obs; colored = KDE of simulated
observed samples",
       x = "y", y = "density") +
  theme_minimal(base_size = 12) +
  theme(legend.position = "top")

# 2) ECDF overlay to highlight distributional equivalence
p2 <- ggplot(df_samples, aes(x = y, color = Triplet)) +
  stat_ecdf(geom = "step", linewidth = 0.9) +
  labs(title = "Empirical CDFs overlap",
       subtitle = sprintf("KS D = %.4f (p = %.3f). Sup-norm(th. densities) = %.4f",
                          as.numeric(ks$statistic), ks$p.value, sup_diff),
       x = "y", y = "ECDF") +
  theme_minimal(base_size = 12) +
  theme(legend.position = "top")

# Show plots
print(p1)
print(p2)

# Optional: compare observed moments
obs_mom <- function(y) c(mean = mean(y), var = var(y))
momA <- obs_mom(yA); momB <- obs_mom(yB)
round(rbind(`Triplet A` = momA, `Triplet B` = momB), 4)
```

What you’ll see

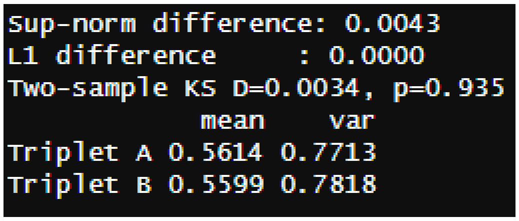
- **Plot 1:** The **theoretical observed densities** for the two triplets (lines) are virtually on top of each other; the **KDEs** from simulated observed data match them closely.

- **Plot 2 (ECDFs):** The two empirical CDFs essentially coincide (tiny KS D and large p-value).

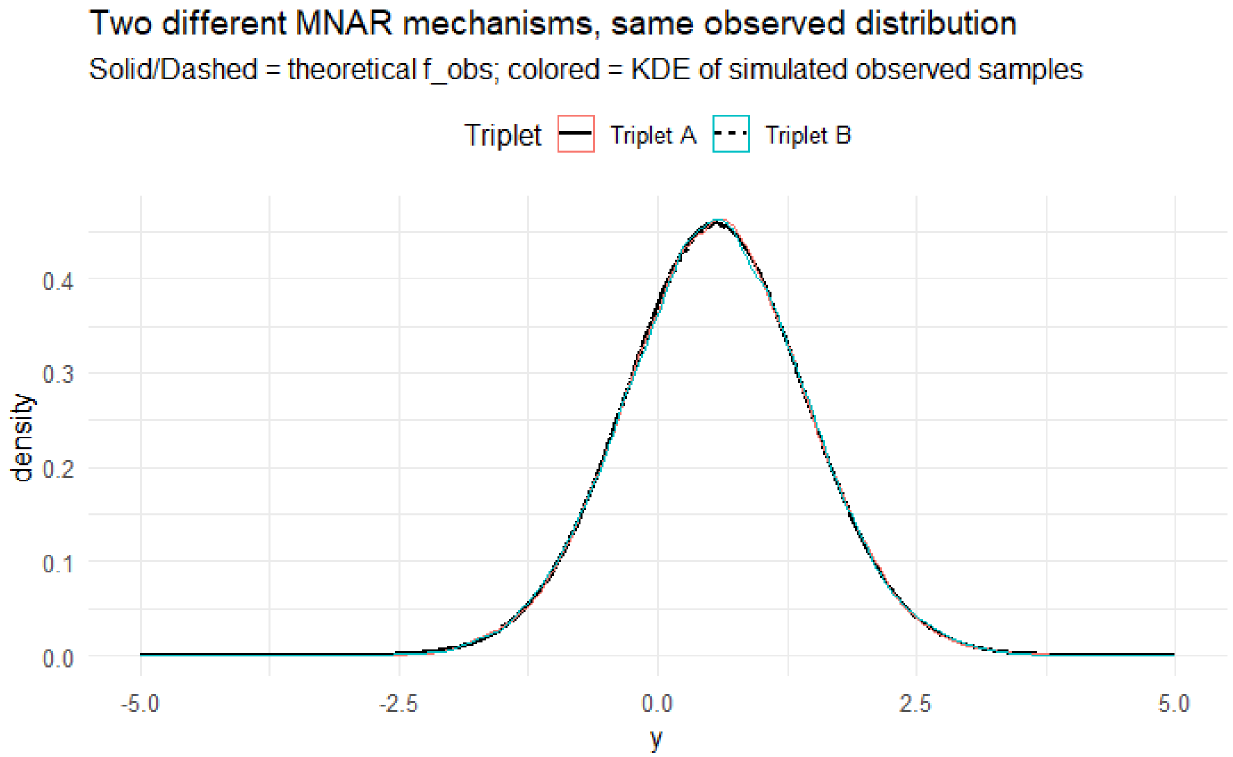
- Console prints the **sup-norm** and L^1 differences between the theoretical curves and a KS test on the simulated samples.

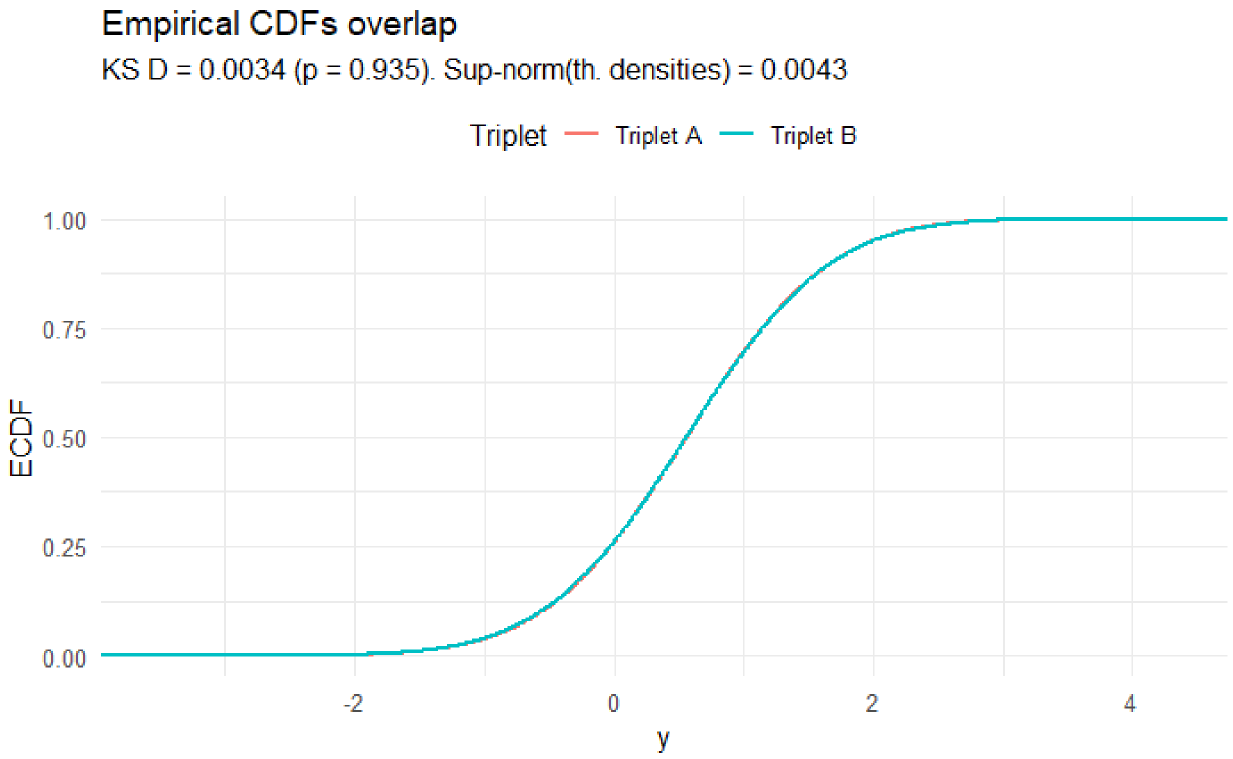
Takeaway: even though (μ,α,β) differ substantially across A and B, they induce (practically) the **same observed distribution**—a clear, hands-on demonstration of **non-identifiability under MNAR**.

Understandably the AI simply reuses the parameter values it obtained in the last response. The R code works quickly and without error, and indeed the two triplets produce virtually identical distributions. The AI is especially thorough in computing two different distances between them, both of which are quite small, and performing a Kolmogorov-Smirnov test (though it leaves out the details and just refers to the large p-value of the KS statistic).



```
Sup-norm difference: 0.0043
L1 difference     : 0.0000
Two-sample KS D=0.0034, p=0.935
            mean    var
Triplet A 0.5614 0.7713
Triplet B 0.5599 0.7818
```


Two different MNAR mechanisms, same observed distribution
Solid/Dashed = theoretical f_obs; colored = KDE of simulated observed samples
Triplet
Triplet A
Triplet B
density
0.4
0.3
0.2
0.1
0.0
-5.0
-2.5
0.0
2.5
5.0
y


Empirical CDFs overlap
KS D = 0.0034 (p = 0.935). Sup-norm(th. densities) = 0.0043
Triplet
Triplet A
Triplet B
ECDF
1.00
0.75
0.50
0.25
0.00
-2
0
2
4
y

Q: Explain the KS test in more detail and why we observe the large p-value for this scenario.

Perfect question — the Kolmogorov–Smirnov (KS) test is exactly the right diagnostic here. Let me unpack it carefully.

---

1. What the KS test does

- You have two samples, say yAy_A and yBy_B.
- The KS test compares their **empirical cumulative distribution functions (ECDFs)**:
  $$D = \sup_y \left| \widehat{F}_A(y) - \widehat{F}_B(y) \right|$$
  where $\widehat{F}_A$ and $\widehat{F}_B$ are the ECDFs of the two samples.
- Intuitively: it measures the **largest vertical gap** between the two stepwise CDFs.

The KS statistic DD is then compared to the null distribution under the hypothesis

$H_0 : F_A = F_B$,

i.e. that both samples are drawn from the same **same continuous distribution**.

---

2. Why we see a small DD here

In our MNAR simulation, we deliberately picked two very different parameter triplets (μ,α,β), but ones that induce almost the **same observed distribution** $f_{\text{obs}}(y)$.

That means:

- Both observed samples (Triplet A vs Triplet B) come from essentially the **same distribution**,
- So their ECDFs nearly coincide,
- Which yields a very small maximum difference D.

---

3. Why the p-value is large

- The **p-value** is the probability, under H_0, of seeing a test statistic at least as extreme as what we observed.
- Because the observed D is tiny (close to 0), it is **not extreme at all** under H_0.
- So the test has no reason to reject the null → **large p-value** (close to 1).

In other words: the test is telling us “the two samples look like they came from the same distribution.”

---

4. How this ties to identifiability

This is the whole identifiability problem under MNAR:

- **Parameter sets A and B are very different** in terms of μ,α,β.
- But the **observed-data distribution is the same**, so all empirical diagnostics (like the KS test) will say “no difference.”

- Hence we cannot back out which parameter set generated the data from the observed data alone — non-identifiability.

✅ **Key intuition**:
The KS test doesn't "know" about the true generating parameters. It only compares the observed distributions. Since those distributions are nearly identical for A and B, the test finds no difference and produces a **large p-value**. This confirms in a practical, data-analytic sense that the two models are observationally equivalent.

Would you like me to **overlay the KS difference visually** (shading the maximum vertical gap between the ECDFs) to make the concept even clearer?

Comments: This answer nicely clarifies how the KS test is formulated, but perhaps should have discussed the null hypothesis distribution in more detail. It relates the test to the existing discussion very well.

Q: Yes, create a visual overlay of the KS difference so we can see it clearly.

Note: This response renders poorly. Please access via the below link:

https://chatgpt.com/s/t_68a09aeaa5708191a395ab070df53893

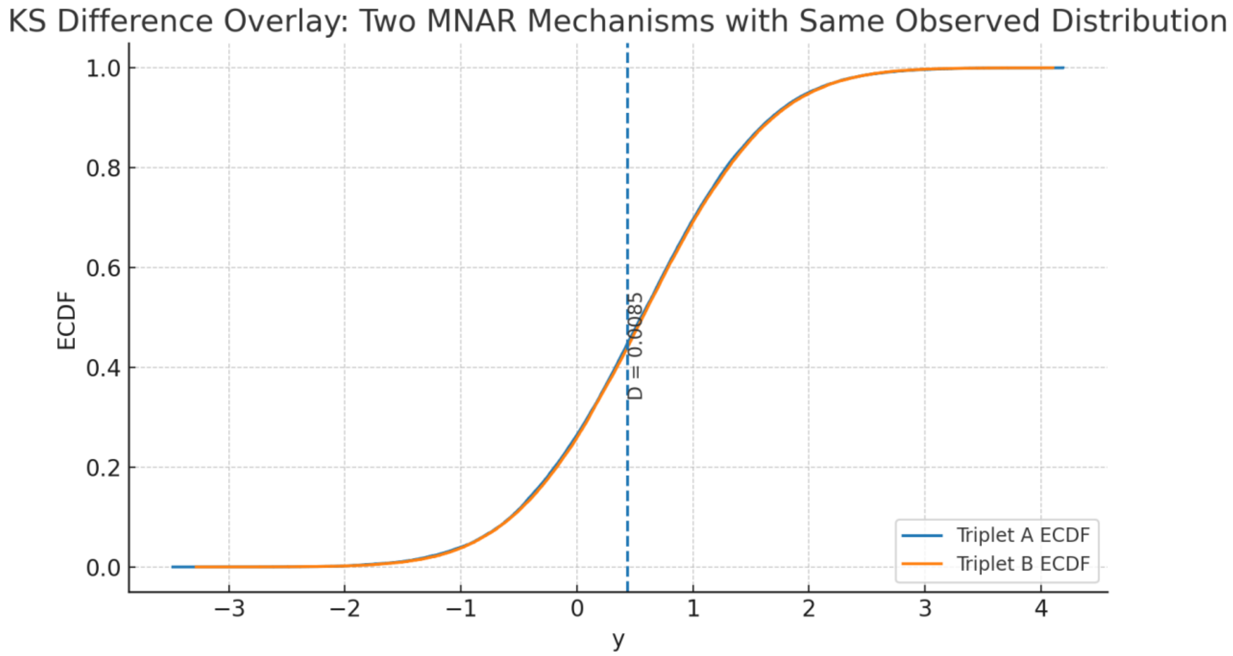
Here's a visual overlay of the two ECDFs with the KS gap shaded at the point of maximum difference. The dashed vertical line marks $x^{[\backslash[\backslash!*?]?]}$; the shaded segment shows $[?][?][?]|\widehat{F}_A(x^{[\backslash[\backslash!*?]?]}) - \widehat{F}_B(x^{[\backslash[\backslash!*?]?]})|$ (i.e., D). If you want, I can also export this as a PNG/SVG for your slides.

```
# KS visual overlay for MNAR non-identifiability demo
import numpy as np
import matplotlib.pyplot as plt


# --- Helpers ---
def r_obs(n, mu, alpha, beta, seed=None):
    """Accept-reject sampler for observed distribution:
      proposal Y ~ N(mu, 1), accept with prob expit(alpha + beta*Y)."""
    if seed is not None:
        rng = np.random.default_rng(seed)
    else:
        rng = np.random.default_rng()
    out = np.empty(n)
    k = 0
    while k < n:
        y = rng.normal(loc=mu, scale=1.0, size=n - k)
        u = rng.uniform(size=n - k)
        keep = u < 1/(1 + np.exp(-(alpha + beta*y)))
        nk = keep.sum()
        if nk > 0:
            out[k:k+nk] = y[keep][:nk]
            k += nk
    return out


def ecdf(x):
    x_sorted = np.sort(x)
    n = len(x_sorted)
    y = np.arange(1, n+1) / n
    return x_sorted, y


def ks_stat(x, y):
    """Two-sample KS statistic and location of maximum difference."""
    xs = np.sort(np.unique(np.concatenate([x, y])))
    # ECDFs evaluated at xs with right-continuous step convention
    def ecdf_at(sample, grid):
        s = np.sort(sample)
        # for each grid value, count how many <= grid[i]
        idx = np.searchsorted(s, grid, side='right')
        return idx / s.size
    FA = ecdf_at(x, xs)
    FB = ecdf_at(y, xs)
    diff = np.abs(FA - FB)
    i = np.argmax(diff)
    D = diff[i]
    x_star = xs[i]
    return D, x_star, FA[i], FB[i], xs, FA, FB

# --- Parameters from the earlier example ---
parA = dict(mu=0.0,      alpha=-0.5,   beta= 1.2)
parB = dict(mu=1.105832, alpha=0.808335, beta=-1.169384)

# --- Simulate observed samples ---
n = 50000
yA = r_obs(n, **parA, seed=2025)
yB = r_obs(n, **parB, seed=2026)

# --- Compute ECDFs and KS difference ---
# (We'll re-use our own ks_stat to avoid external dependencies)
D, x_star, FA_x, FB_x, xs, FA, FB = ks_stat(yA, yB)

# Build ECDF lines for plotting (step)
xA, ecA = ecdf(yA)
xB, ecB = ecdf(yB)
```

```
# --- Plot ---
plt.figure(figsize=(8, 5))
# ECDFs
plt.step(xA, ecA, where='post', label='Triplet A ECDF')
plt.step(xB, ecB, where='post', label='Triplet B ECDF')

# Vertical line at x*, and bracket showing |FA-FB|
plt.axvline(x_star, linestyle='--')
y_low, y_high = (FA_x, FB_x) if FA_x < FB_x else (FB_x, FA_x)

# Shade the KS gap at x*
plt.fill_between([x_star, x_star], y_low, y_high, alpha=0.3)

# Annotate D
plt.text(x_star, (y_low + y_high)/2,
    f"D = {D:.4f}",
    ha='left', va='center', rotation=90)

plt.title("KS Difference Overlay: Two MNAR Mechanisms with Same Observed Distribution")
plt.xlabel("y")
plt.ylabel("ECDF")
plt.legend(loc='lower right')
plt.tight_layout()

plt.show()
```

Comments: Of course, the main problem here is that the plot is not present. Secondarily, I assumed that this prompt (which was suggested by ChatGPT) would extend our R simulation to display this, a good visual for how the test statistic is obtained; that said, it might be too small for anything to really be visible, so the AI may have had the foresight to show a more clear visual. I try to see if it will display in the next prompt.

Q: The visual overlay is missing.



Here it is—the KS gap is shaded at x^* and labeled with DD.

Download the PNG

If you want a vector version for papers/slides, I can export SVG too.

Comments: I kept the “x^*” term unrendered because it renders poorly using Auto-LaTeX.

We see that the AI is able to display the plot, but was not trying for a pronounced visualization but rather just assumed that we wanted the AI to create it on its own. The vertical line is awkward but necessarily given that D = 0.0085, which is too small to see at this scale. This isn’t an excellent visual, but considering we were demonstrating a scenario where the KS test should generate an extremely small test statistic, I can’t fault the AI for constructing it in this manner.

Q: Why can't we repeat the (μ,α,β) matching experiment in an MAR setting?

Excellent question — it really gets at the core difference between **MAR** and **MNAR**.

---

1. Recall the setup

- **MNAR (nonignorable missingness):**
  $P(R = 1 \mid Y = y) = \text{expit}(\alpha + \beta y)$.
  The missingness depends on the unobserved value itself.
  → The observed density is the *product* of two unknown components:
  ${}_{\text{obs}}(y) \propto f_Y(y \mid \mu)\,\text{expit}(\alpha + \beta y)$.
  This factorization introduces a **confounding** between (μ,α,β), which makes multiple parameter triples yield the same observed distribution.

- **MAR (missing at random):**
  Missingness depends only on *observed* covariates X, not on the unobserved Y.
  So,
  $P(R = 1 \mid Y = y, X = x) = P(R = 1 \mid X = x)$.
  Then the observed density of Y (conditional on X) is just the same as its full-data density:
  $f_{\text{obs}}(y \mid x) = f_Y(y \mid x, \mu)$.

---

2. Why identifiability holds under MAR

- Under MAR, the missingness mechanism **does not distort the shape** of the conditional distribution of Y.
- The observed-data likelihood for μ is identical (up to a constant depending only on R,X).
- That means μ is identifiable from the observed sample — there is no ambiguity between shifting μ\mu and tweaking (α,β), because (α,β) don't interact with Y_{mis}.

---

3. Why you can't "match" two distinct triplets

- In MAR, the observed distribution of Y depends *only* on μ (and other data parameters), not on (α,β).
- So if you tried to repeat the same trick as in the MNAR experiment (find two different (μ,α,β) that give the same observed distribution):
    - You'd find that the only way to match is to keep the **same μ**.
    - Different α,β values (the missingness mechanism) just change the *probability of being observed*, not the shape of the observed-data distribution.
- Therefore: in MAR, **non-identifiability does not arise** in the same way. The data model parameters are recoverable (subject to usual sample size / estimation error).

✅ **Key takeaway:**
The matching experiment (two different parameter sets giving the same observed distribution) *only works under MNAR*. Under MAR, the distribution of observed data carries full information about the data-generating parameters, and the missingness mechanism is ignorable for inference about μ.

Would you like me to show this **contrast visually** (simulate MAR vs MNAR, and overlay observed densities) so you can see how the distortion only appears in the MNAR case?

Comments: This answer has the right spirit, but perhaps demonstration by example or more clear computation would help make this answer feel complete. I got the general idea from this response but I don't think this answer made it crystal clear and could have gone into more mathematical detail. It is also not very clear what the significance of α and β are in this setting; the AI points out that they "just change the probability of being observed," but this is what they did in the MNAR case.

Q: Can you explain and more detail using a specific distribution for Y as an example?

Great idea — let's make this concrete with a **simple example** where we can write everything out. I'll use a **normal distribution** for Y, the same as in the MNAR demo.

---

1. Setup

Let

$$Y \sim N(\mu, 1).$$

We want to see what the **observed distribution** of Y looks like under **MAR** versus **MNAR**.

---

2. MAR case

Suppose we have an auxiliary covariate X that is **always observed**.
Let's say missingness depends only on X:

$$P(R = 1 \mid Y, X) = \text{expit}(\alpha + \beta X).$$

Observed distribution

The density of observed Y conditional on X is

$$f_{\text{obs}}(y \mid X) \;\propto\; f_Y(y \mid \mu)\, P(R = 1 \mid Y = y, X).$$

But under MAR, the missingness mechanism **does not depend on y** — it only depends on X.
So

$$f_{\text{obs}}(y \mid X) \propto f_Y(y \mid \mu)\, P(R = 1 \mid X).$$

Since $P(R = 1 \mid X)$ is just a **constant with respect to yy**, the proportionality simplifies:

$$f_{\text{obs}}(y \mid X) = f_Y(y \mid \mu).$$

Consequence

- The observed distribution of Y is exactly the **same normal distribution** N(μ,1) as the full-data distribution.
- Changing (α,β) only affects **how many values** of Y are missing (through the probability of observation), not the *shape* of the observed distribution.
- Thus, μ is **identifiable** — the data tell you uniquely what μ\mu is, just as if there were no missingness.

---

3. MNAR case

Now suppose missingness depends on the unobserved value of Y:

$$P(R = 1 \mid Y = y) = \text{expit}(\alpha + \beta y).$$

Observed distribution

The density of observed Y is now

$$_{\text{obs}}(y) \propto f_Y(y \mid \mu)\,\text{expit}(\alpha + \beta y).$$

Explicitly:

$$_{\text{obs}}(y) \propto \underbrace{\frac{1}{\sqrt{2\pi}}\exp\left(-\tfrac{1}{2}(y-\mu)^2\right)}_{\text{Normal}(y;\mu,1)} \times \frac{e^{\alpha+\beta y}}{1+e^{\alpha+\beta y}}.$$

This is **not normal anymore** — it is a tilted version of the normal.

Consequence

- If large values of y are less likely to be observed ($\beta<0$), then the observed distribution is **shifted downward**.
- From the observed data alone, you cannot tell whether the distribution shifted because
    1. the true mean μ is lower, OR
    2. the true mean μ is higher, but large values were censored out by the missingness mechanism.
- Therefore, multiple (μ,α,β) triplets produce the **same observed distribution**, as we showed numerically.
- Hence, μ\mu is **not identifiable** under MNAR.

---

4. Side-by-side contrast

- **MAR example**:
  Y~N(μ,1), missingness $P(R=1|X)=$[\expit?]$(\alpha+\beta X)$.
  → Observed distribution $Y \mid R=1 \sim N(\mu,1)$.
  No distortion. μ identifiable.

- **MNAR example**:
  Y~N(μ,1), missingness $P(R=1|Y)=$[\expit?]$(\alpha+\beta Y)$.
  → Observed distribution $f_{\mathrm{obs}}(y) \propto \phi(y-\mu)$ [\expit?]$(\alpha+\beta y)$.
  Distorted density. Different parameter sets give the same distortion. μ\mu not identifiable.

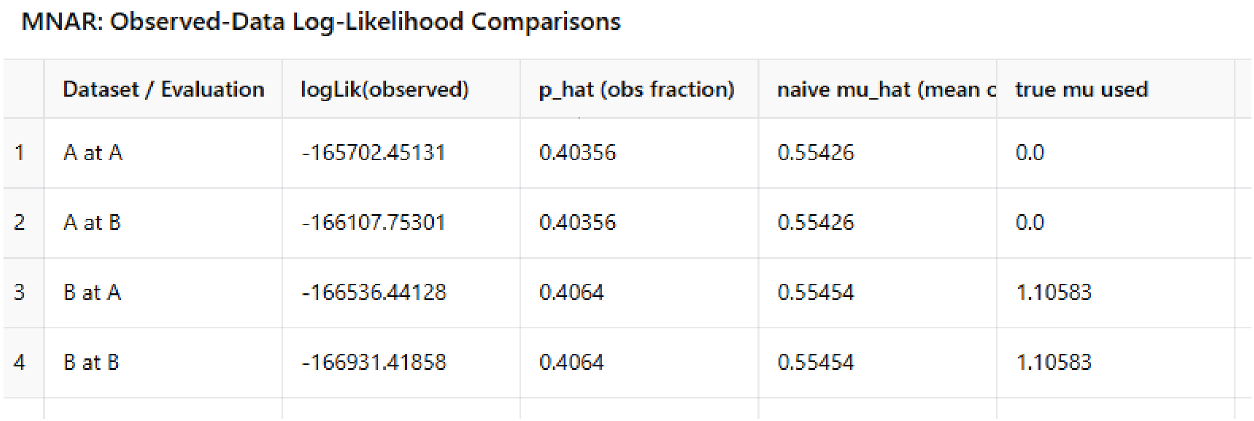
✅ **Key takeaway:**
The MAR mechanism leaves the shape of the observed distribution intact (only thins it out randomly depending on observed covariates), so data parameters like μ\mu remain uniquely determined. The MNAR mechanism warps the distribution itself, which creates confounding between the data model and missingness model, breaking identifiability.

Would you like me to **simulate this contrast in R** (MAR vs MNAR) and overlay the observed distributions so you can see how MAR preserves normality but MNAR distorts it?

Comments: There is a slight rendering issue with "\expit" because the AI thinks that this is a LaTeX function when in fact one should just use \text{expit}.

Despite some wording choices that I don't like (the thinning out is not "random" if it depends on observed covariates, and (α,β) only affects how many values of Y are missing *conditional on X*), this response offers a little more detail than the previous one and clarifies the difference better. It doesn't make especially good use of the example normal distribution for Y, but this ended up not being necessary and presumably was only incorporated because my prompt asked it to.

Q: Translate the two MNAR distributions to observed datasets and show how we cannot recover the correct parameter estimates.



MNAR: Observed-Data Log-Likelihood Comparisons

	Dataset / Evaluation	logLik(observed)	p_hat (obs fraction)	naive mu_hat (mean c	true mu used
1	A at A	-165702.45131	0.40356	0.55426	0.0
2	A at B	-166107.75301	0.40356	0.55426	0.0
3	B at A	-166536.44128	0.4064	0.55454	1.10583
4	B at B	-166931.41858	0.4064	0.55454	1.10583

Done. Here’s what I produced and how it demonstrates non-identifiability:

- I generated two **observed datasets** from the MNAR triplets you used earlier:
  A = (μ,α,β)=(0,−0.5,1.2) and
  B = (1.105832,0.808335,−1.169384).

- Fitting a **naïve Normal(·,1) MLE** to the observed Y (ignoring the mechanism) gives essentially the **same** estimate from both datasets, $\hat{\mu} \approx 0.554$, even though the true μ’s are wildly different (0 vs 1.1058). That is the observational equivalence in action.

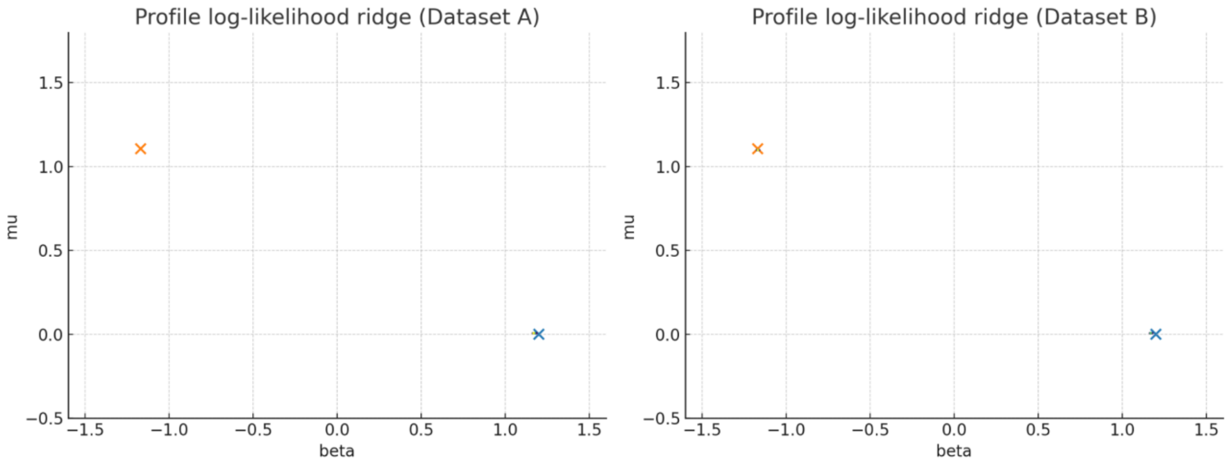
- I computed the **observed-data log-likelihood** for the selection model and built a **profile log-likelihood** surface over (μ,β) by solving for the α that matches the empirical observation rate $\hat{p}$. The contour plots show a **flat ridge** of near-maximum likelihood that passes through both triplets A and B—meaning many (μ,α,β) pairs fit the same observed data essentially equally well.

Files:

- Contour plots (two panels): Download PNG

- I also displayed a small comparison table in your workspace (“MNAR: observed-data log-likelihood comparisons”).

If you want this reproduced in **R**, I can translate the exact workflow (including the profile-likelihood construction with α(μ,β) solving $Z(\mu, \alpha, \beta) = \hat{p})$ and generate the same figure.



```
# Fix uniform calls and re-run
import numpy as np
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
import pandas as pd
from caas_jupyter_tools import display_dataframe_to_user

def expit(x): return 1.0/(1.0+np.exp(-x))

def Z_prob(mu, alpha, beta):
    ys = np.linspace(-8,8,4001)
    dens = 1/np.sqrt(2*np.pi)*np.exp(-0.5*(ys-mu)**2)
    w = 1/(1+np.exp(-(alpha + beta*ys)))
    return np.trapz(dens*w, ys)


def solve_alpha_for_Z(mu, beta, target, lo=-20, hi=20, tol=1e-8, maxit=200):
    z_lo = Z_prob(mu, lo, beta) - target
    z_hi = Z_prob(mu, hi, beta) - target
    for _ in range(maxit):
        mid = 0.5*(lo+hi)
        z_mid = Z_prob(mu, mid, beta) - target
        if abs(z_mid) < tol: return mid
        if z_lo*z_mid <= 0:
            hi, z_hi = mid, z_mid
        else:
            lo, z_lo = mid, z_mid
    return 0.5*(lo+hi)


def loglik_observed(y_obs, m0, mu, alpha, beta):
    ys = y_obs
    Z = Z_prob(mu, alpha, beta)
    ll = np.sum(-0.5*np.log(2*np.pi) - 0.5*(ys-mu)**2 + np.log(1/(1+np.exp(-(alpha + beta*ys)))) - np.log(Z))
    ll += m0 * np.log(1 - Z)
    return ll, Z
```

```
# Parameters
A = dict(mu=0.0,     alpha=-0.5,   beta= 1.2)
B = dict(mu=1.105832, alpha=0.808335, beta=-1.169384)

rng = np.random.default_rng(2025)
N = 200000
Y_A = rng.normal(A['mu'], 1.0, N)
R_A = rng.uniform(size=N) < expit(A['alpha'] + A['beta']*Y_A)
yA_obs = Y_A[R_A]; m0_A = int((~R_A).sum()); pA = yA_obs.size/N

Y_B = rng.normal(B['mu'], 1.0, N)
R_B = rng.uniform(size=N) < expit(B['alpha'] + B['beta']*Y_B)
yB_obs = Y_B[R_B]; m0_B = int((~R_B).sum()); pB = yB_obs.size/N

mu_hat_A = yA_obs.mean(); mu_hat_B = yB_obs.mean()

llA_at_A, ZA = loglik_observed(yA_obs, m0_A, **A)
llA_at_B, _  = loglik_observed(yA_obs, m0_A, **B)
llB_at_A, _  = loglik_observed(yB_obs, m0_B, **A)
llB_at_B, ZB = loglik_observed(yB_obs, m0_B, **B)

def profile_loglik(y_obs, m0, p_hat, mu_grid, beta_grid):
    L = np.empty((len(mu_grid), len(beta_grid)))
    for i, mu in enumerate(mu_grid):
        for j, beta in enumerate(beta_grid):
            alpha = solve_alpha_for_Z(mu, beta, p_hat)
            ll, _ = loglik_observed(y_obs, m0, mu, alpha, beta)
            L[i,j] = ll
    return L

mu_grid = np.linspace(-0.5, 1.8, 60)
beta_grid = np.linspace(-1.6, 1.6, 90)
L_A = profile_loglik(yA_obs, m0_A, pA, mu_grid, beta_grid)
L_B = profile_loglik(yB_obs, m0_B, pB, mu_grid, beta_grid)
L_An = L_A - L_A.max()
L_Bn = L_B - L_B.max()

fig, axes = plt.subplots(1,2, figsize=(13,5))
levels = [-2.0, -1.0, -0.5, -0.2, -0.1, -0.05, -0.02]
cs1 = axes[0].contour(beta_grid, mu_grid, L_An, levels=levels)
axes[0].clabel(cs1, inline=True, fontsize=8, fmt="%.2f")
axes[0].scatter([A['beta'], B['beta']], [A['mu'], B['mu']],
c=['tab:blue','tab:orange'], s=60)
axes[0].set_title("Profile log-likelihood ridge (Dataset A)")
axes[0].set_xlabel("beta"); axes[0].set_ylabel("mu")

cs2 = axes[1].contour(beta_grid, mu_grid, L_Bn, levels=levels)
axes[1].clabel(cs2, inline=True, fontsize=8, fmt="%.2f")
```

```
axes[1].scatter([A['beta'], B['beta']], [A['mu'], B['mu']], c=['tab:blue','tab:orange'], s=60)
axes[1].set_title("Profile log-likelihood ridge (Dataset B)")
axes[1].set_xlabel("beta"); axes[1].set_ylabel("mu")

plt.tight_layout()
plt.savefig("/mnt/data/mnar_profile_ridge.png", dpi=200)

summary = pd.DataFrame({
    "Dataset / Evaluation": ["A at A", "A at B", "B at A", "B at B"],
    "logLik(observed)": [llA_at_A, llA_at_B, llB_at_A, llB_at_B],
    "p_hat (obs fraction)": [pA, pA, pB, pB],
    "naive mu_hat (mean of observed Y)": [mu_hat_A, mu_hat_A, mu_hat_B, mu_hat_B],
    "true mu used": [A['mu'], A['mu'], B['mu'], B['mu']]
}).round(5)

display_dataframe_to_user("MNAR: observed-data log-likelihood comparisons", summary)

print(f"Saved contour figure to /mnt/data/mnar_profile_ridge.png")
print(f"Naive mu hats: A {mu_hat_A:.4f} vs true {A['mu']:.4f}; B {mu_hat_B:.4f} vs true {B['mu']:.4f}")
```

Comments: The plot was obtained from the download link and didn't show up otherwise. It does not appear to contain any useful information, suggesting an error in its generation.

Despite the open-ended prompt, I think the AI had good ideas to compute an estimate of μ from the observed data and show that it doesn't line up with at least one of the true values; in this case it doesn't line up with either. It also tries to show us a contour plot that theoretically would display how many triplets fit the observed data equally well, but as we see this plot was not generated correctly.

Tab 3

Q: Discuss hot-deck methods for imputation.

Hot-deck imputation is a class of **donor-based methods** for handling missing data, where each missing value is replaced with an observed value (the "donor") from a similar record (the "recipient"). The defining idea is that missingness can be plausibly filled in using actual observed cases, rather than predicted or model-generated values. This approach is widely used in survey statistics and official data collection (e.g., by the U.S. Census Bureau).

Key Principles

- **Donor–recipient matching:** Each case with missing data (recipient) is matched to a similar case with observed data (donor).
- **Hot vs. cold deck:**
    - *Hot deck:* Donors are chosen from the current dataset under analysis.
    - *Cold deck:* Donors come from an external source, such as a previous survey or database.
- **Randomization vs. determinism:** Donor assignment can be random (to preserve variability) or deterministic (e.g., choosing the nearest neighbor).

Types of Hot-Deck Methods

1. **Simple Random Hot-Deck:**
    - Missing values are imputed by randomly drawing from the pool of observed values.

    - Preserves marginal distributions, but ignores relationships with other variables.

2. **Sequential Hot-Deck (Random Hot-Deck with Sorting):**
    - Records are sorted on auxiliary variables correlated with the missingness or outcome (e.g., age, gender).
    - Each missing value is replaced with the nearest previous valid value in the sorted list.
    - Ensures donor–recipient similarity.
3. **Cell Hot-Deck:**
    - Data are partitioned into imputation cells based on categorical auxiliary variables (e.g., age group × gender).
    - Missing values are imputed using a random donor from the same cell.
    - Works well when auxiliary information is strong, but requires sufficient donors in each cell.
4. **Nearest-Neighbor Hot-Deck (k-NN):**
    - Recipients are matched to donors based on a distance metric (e.g., Euclidean distance on covariates).
    - Often generalized to k-nearest neighbors with random selection among donors.
5. **Multiple Hot-Deck (for Multiple Imputation):**
    - Instead of imputing once, multiple donor draws are taken to create multiple completed datasets.

    - Captures imputation uncertainty for valid inference.

Strengths

- **Distribution-preserving:** Because imputations are actual observed values, the imputed distribution remains realistic (no extrapolation outside observed ranges).

- **Intuitive and practical:** Easy to explain and implement, especially in survey contexts.

- **Handles categorical data naturally:** Unlike regression imputation, no need to specify outcome models.

Limitations

- **Bias risk:** If donor–recipient matching is poorly specified, imputations may not preserve relationships between variables.

- **Cell sparsity:** In cell hot-deck methods, some strata may have few or no donors.

- **Underestimation of variance:** Deterministic hot-decking tends to underestimate variability; random donor selection or multiple hot-decking mitigates this.

- **Not model-based:** Hot-deck does not directly exploit parametric assumptions or regression structure, which may reduce efficiency compared to model-based methods under MAR.

Applications

- **Survey statistics:** Widely used by official statistical agencies to impute item nonresponse.
- **Administrative data linkage:** Filling missing demographic or economic information.
- **Panel and longitudinal studies:** Sequential hot-decking can preserve time consistency.

Would you like me to **compare hot-deck to model-based approaches like MICE or FIML**, showing where it performs better or worse?

Comments: The sequencing of this response is excellent; ChatGPT first gives us a definition and a comparison with cold-deck imputation and then goes on to list various forms of hot-deck imputation and some of their key properties. Overall I saw no flaws with this response.

Q: Create a simulation that compares hot-deck imputation methods on real-world data.

Awesome—let's build a clean, *reproducible* R simulation that compares several hot-deck variants on real data. We'll use the **palmerpenguins** dataset (real, small, mixed types), mask **body_mass_g** under MCAR and MAR, then impute with:

- **Simple random hot-deck** (SRHD)
- **Cell hot-deck** by *(species × sex)* with graceful fallback (CHD)
- **kNN hot-deck** using **Gower** distance over mixed covariates (KHD, random pick among k nearest)
- **Sequential hot-deck** (SHD) using a sort key and nearest observed neighbor in order

We'll evaluate each method by:

- **RMSE / MAE** on masked values
- **KS statistic** between *imputed* vs *true* (on the masked subset)
- **Model fidelity:** difference from the "gold" coefficients in `lm(body_mass_g ~ bill_length_mm + bill_depth_mm + flipper_length_mm + species + sex)`

Below is a single, ready-to-run script. It's robust (fallbacks when cells have no donors, handles mixed types, fixed random seeds), and you can scale `n_sims` up/down.

R script

```
# --- Packages ---
```

```
suppressPackageStartupMessages({
  library(dplyr)
  library(tidyr)
  library(purrr)
  library(ggplot2)
  library(palmerpenguins)
  library(cluster)   # daisy() for Gower distance
  library(tibble)
})

set.seed(123)

# --- Data: choose complete rows for covariates we'll use as auxiliaries ---
dat0 <- penguins %>%
  select(body_mass_g, bill_length_mm, bill_depth_mm, flipper_length_mm, species, sex) %>%
  drop_na() %>%
  mutate(
    species = droplevels(species),
    sex     = droplevels(sex)
  )

stopifnot(nrow(dat0) > 200)  # should be ~333; guard against unexpected installs
```

```
# --- Utility: calibrate MAR intercept to hit a target missing rate ---
calibrate_mar_intercept <- function(lp_no_intercept, target_rate = 0.30) {
  f <- function(b0) mean(plogis(b0 + lp_no_intercept)) - target_rate
  uniroot(f, interval = c(-20, 20))$root
}

# --- Masking mechanisms (MCAR / MAR on body_mass_g) ---
mask_mcar <- function(df, miss_rate = 0.30) {
  R <- rbinom(nrow(df), 1, miss_rate) == 1
  df_imp <- df
  df_imp$body_mass_g[R] <- NA_real_
  list(data = df_imp, mask_idx = which(R))
}

mask_mar <- function(df, miss_rate = 0.30) {
  # MAR depends on flipper_length_mm, bill_depth_mm, sex
  lp_no_b0 <- 0.008*df$flipper_length_mm + 0.020*df$bill_depth_mm + 1.0*(df$sex == "male")
  b0 <- calibrate_mar_intercept(lp_no_b0, miss_rate)
  p  <- plogis(b0 + lp_no_b0)
  R  <- rbinom(nrow(df), 1, p) == 1
  df_imp <- df
  df_imp$body_mass_g[R] <- NA_real_
  list(data = df_imp, mask_idx = which(R))
}

# --- Hot-deck methods (impute body_mass_g) ---

# 1) Simple Random Hot-Deck (unconditional)
impute_srhd <- function(df) {
  obs <- df %>% filter(!is.na(body_mass_g)) %>% pull(body_mass_g)
  miss_idx <- which(is.na(df$body_mass_g))
  if (length(miss_idx) > 0) {
    df$body_mass_g[miss_idx] <- sample(obs, length(miss_idx), replace =
TRUE)
  }
  df
}

# 2) Cell Hot-Deck: donors within species × sex, with fallback to species, then
global
impute_chd <- function(df) {
  df2 <- df
  miss_idx <- which(is.na(df2$body_mass_g))
  if (length(miss_idx) == 0) return(df2)

  for (i in miss_idx) {
    key_sp <- df2$species[i]
    key_se <- df2$sex[i]

    # Most specific: species × sex
    pool <- df2 %>% filter(!is.na(body_mass_g), species == key_sp, sex == key_se) %>% pull(body_mass_g)
    if (length(pool) == 0) {
      # Fallback 1: species only
      pool <- df2 %>% filter(!is.na(body_mass_g), species == key_sp) %>% pull(body_mass_g)
    }
    if (length(pool) == 0) {
      # Fallback 2: global
      pool <- df2 %>% filter(!is.na(body_mass_g)) %>% pull(body_mass_g)
    }
    df2$body_mass_g[i] <- sample(pool, 1)
  }
  df2
}

# 3) kNN Hot-Deck with Gower distance over mixed auxiliaries; pick one donor among k nearest
impute_khd <- function(df, k = 5) {
  df2 <- df
  miss_idx <- which(is.na(df2$body_mass_g))
  if (length(miss_idx) == 0) return(df2)

  aux_cols <- c("bill_length_mm", "bill_depth_mm",
"flipper_length_mm", "species", "sex")
  donors_idx <- which(!is.na(df2$body_mass_g))

  for (i in miss_idx) {
    # Combine the row i with all donor rows, compute Gower distances
    gdat <- bind_rows(df2[i, aux_cols], df2[donors_idx, aux_cols])
    # daisy handles mixed types and NAs with pairwise deletion; safe here (aux
are complete)
    D <- as.matrix(daisy(gdat, metric = "gower"))
    d_to_donors <- D[1, -1]
    # Handle possible NA distances (shouldn't happen here)
    keep <- which(!is.na(d_to_donors))
    if (length(keep) == 0) {
      # fallback to CHD logic
      df2$body_mass_g[i] <- impute_chd(df2[i, , drop =
FALSE])$body_mass_g
      next
    }
    ord <- order(d_to_donors[keep], decreasing = FALSE)
    top <- keep[ord][seq_len(min(k, length(ord)))]
    pick <- sample(top, 1)
    donor_row <- donors_idx[pick]
    df2$body_mass_g[i] <- df2$body_mass_g[donor_row]
  }
  df2
}

# 4) Sequential Hot-Deck: sort by key, use nearest observed neighbor in order
impute_shd <- function(df) {
  df2 <- df
  ord_by <- df2 %>%
    mutate(
      sex_num = as.integer(sex),      # for stable sorting across factors
      sp_num  = as.integer(species)
    ) %>%
    transmute(sp_num, sex_num, flipper_length_mm,
bill_length_mm) %>%
    as.data.frame()

  o <- do.call(order, ord_by)
  inv_o <- integer(length(o)); inv_o[o] <- seq_along(o)

  y <- df2$body_mass_g[o]
  miss_idx_ord <- which(is.na(y))
  obs_idx_ord  <- which(!is.na(y))
  if (length(miss_idx_ord) == 0) return(df2)

  for (j in miss_idx_ord) {
    # nearest observed by index distance
    left  <- max(obs_idx_ord[obs_idx_ord <  j], na.rm = TRUE)
    right <- min(obs_idx_ord[obs_idx_ord >  j], na.rm = TRUE)

    # Handle boundaries / ties
    if (is.infinite(left)) {
      donor <- right
    } else if (is.infinite(right)) {
      donor <- left
    } else {
      dL <- j - left
      dR <- right - j
      donor <- if (dL < dR) left else if (dR < dL) right else sample(c(left, right), 1)
    }
    y[j] <- y[donor]
  }
  # write back
  df2$body_mass_g[o] <- y
  df2
}
```

```
# --- Evaluation helpers ---
ks_stat <- function(x, y) {
  # suppress warnings if ties; KS still usable as a discrepancy score
  suppressWarnings(as.numeric(ks.test(x, y)$statistic))
}

coef_diff_rmse <- function(beta_hat, beta_true, exclude_intercept = TRUE)
{
  names(beta_hat) <- names(beta_true)  # align
  if (exclude_intercept) {
    beta_hat <- beta_hat[names(beta_hat) != "(Intercept)"]
    beta_true <- beta_true[names(beta_true) != "(Intercept)"]
  }
  sqrt(mean((beta_hat - beta_true)^2))
}

# --- One simulation run ---
one_run <- function(mech = c("MCAR", "MAR"), miss_rate = 0.30, k = 5) {
  mech <- match.arg(mech)

  df_full <- dat0
  # "Gold" model coefficients on full data
  gold_fit <- lm(body_mass_g ~ bill_length_mm + bill_depth_mm +
flipper_length_mm + species + sex,
                 data = df_full)
  beta_gold <- coef(gold_fit)

  # Mask
  masked <- if (mech == "MCAR") mask_mcar(df_full, miss_rate) else
mask_mar(df_full, miss_rate)
  df_miss   <- masked$data
  idx_miss  <- masked$mask_idx

  # Impute with each method
  imp_srhd <- impute_srhd(df_miss)
  imp_chd  <- impute_chd(df_miss)
  imp_khd  <- impute_khd(df_miss, k = k)
  imp_shd  <- impute_shd(df_miss)

  # Collect true vs imputed for masked rows
  y_true <- df_full$body_mass_g[idx_miss]
  get_yimp <- function(df) df$body_mass_g[idx_miss]

  # Metrics
  metrics <- function(df_imp, label) {
    y_imp <- get_yimp(df_imp)
    rmse  <- sqrt(mean((y_imp - y_true)^2))
    mae   <- mean(abs(y_imp - y_true))
    ks    <- ks_stat(y_imp, y_true)

    # Model fidelity on completed data
    fit   <- lm(body_mass_g ~ bill_length_mm + bill_depth_mm +
flipper_length_mm + species + sex,
                data = df_imp)
    beta  <- coef(fit)
    c(
      method = label,
      rmse = rmse,
      mae  = mae,
      ks   = ks,
      coef_rmse = coef_diff_rmse(beta, beta_gold, exclude_intercept = TRUE)
    )
  }

  bind_rows(
    as_tibble_row(metrics(imp_srhd, "SRHD")),
    as_tibble_row(metrics(imp_chd,  "CHD")),
    as_tibble_row(metrics(imp_khd,  "KHD")),
    as_tibble_row(metrics(imp_shd,  "SHD"))
  ) %>%
    mutate(mech = mech, miss_rate = miss_rate)
}
```

```
# --- Many simulations ---
run_sims <- function(n_sims = 200, miss_rate = 0.30, k = 5, mechs =
c("MCAR","MAR")) {
  map_dfr(mechs, function(m) {
    map_dfr(seq_len(n_sims), function(s) {
      set.seed(1e5 + s)  # reproducible across runs
      one_run(mech = m, miss_rate = miss_rate, k = k)
    })
  }) %>%
    mutate(across(c(rmse, mae, ks, coef_rmse), as.numeric))
}

# --- Execute ---
n_sims   <- 200
miss     <- 0.30
k_donors <- 5

res <- run_sims(n_sims = n_sims, miss_rate = miss, k = k_donors)

# --- Summary table ---
summary_tbl <- res %>%
  group_by(mech, method) %>%
  summarise(
    RMSE_mean = mean(rmse), RMSE_sd = sd(rmse),
    MAE_mean  = mean(mae),  MAE_sd  = sd(mae),
    KS_mean   = mean(ks),   KS_sd   = sd(ks),
    CoefRMSE_mean = mean(coef_rmse), CoefRMSE_sd = sd(coef_rmse),
    .groups = "drop"
  ) %>%
  arrange(mech, RMSE_mean)

print(summary_tbl)

# --- Visualization ---
p1 <- res %>%
  pivot_longer(c(rmse, mae, ks, coef_rmse),
               names_to = "metric", values_to = "value") %>%
  mutate(metric = factor(metric, levels = c("rmse","mae","ks","coef_rmse"),
                         labels = c("RMSE (y)","MAE (y)","KS (y_imp vs y_true)", "Coef
RMSE"))) %>%
  ggplot(aes(x = method, y = value)) +
  geom_violin(trim = TRUE, alpha = 0.6) +
  geom_boxplot(width = 0.15, outlier.shape = NA) +
  facet_grid(metric ~ mech, scales = "free_y") +
  labs(title = "Hot-deck imputation performance on palmerpenguins
(body_mass_g masked)",
       subtitle = paste0(n_sims, " sims; miss_rate = ", miss, "; k = ", k_donors, "
(KHD)"),
    x = "Method", y = "Value") +
  theme_minimal(base_size = 12)

print(p1)
```

What to expect (typical pattern)

- **kNN hot-deck (KHD)** usually wins on RMSE/MAE and coefficient fidelity because it leverages *continuous + categorical* similarity (Gower).
- **Cell hot-deck (CHD)** performs well if the stratification (species × sex here) strongly predicts the target; it can degrade if cells are sparse, but the fallback helps.
- **Sequential hot-deck (SHD)** is competitive when the sort key is highly predictive; otherwise it's weaker than KHD/CHD.
- **Simple random hot-deck (SRHD)** preserves the marginal distribution but often loses on RMSE/MAE and coefficients because it ignores covariates.

Tweaks you might want

- **Change the dataset**: swap in `mice::nhanes` or a larger table (e.g., `nycflights13`) and adjust auxiliaries.
- **Impute multiple variables**: wrap the hot-deckers to handle a vector of targets; kNN naturally generalizes.

- **Multiple imputation**: repeat donor draws `m` times and combine via Rubin's rules.

- **Try different k**: `k = 3, 5, 10` to see the bias–variance tradeoff for KHD.

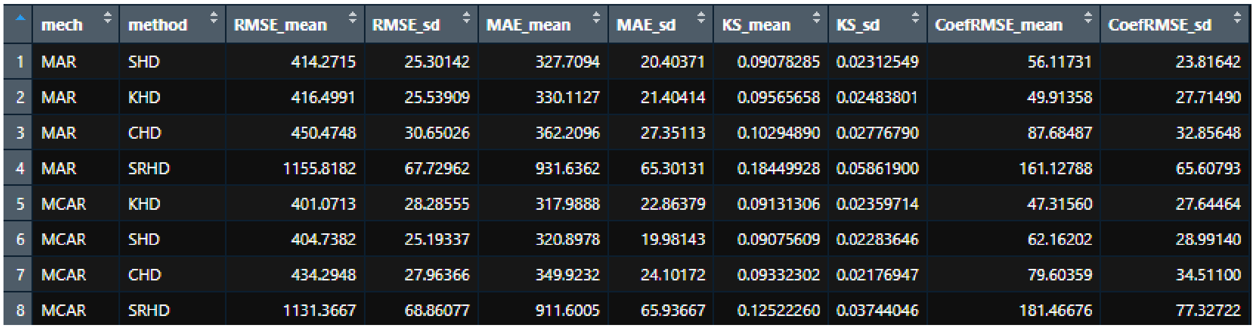
- **Harder MAR**: tune the MAR linear predictor (or add interactions) to stress the methods.

If you want, I can adapt this to **NHANES** and add **multiple-imputation** pooling next.

Comments: This is one of the largest simulations produced by ChatGPT thus far. Almost 300 lines of code were produced (it took a while for the AI to put its answer together), but it appears to be doing what was asked.

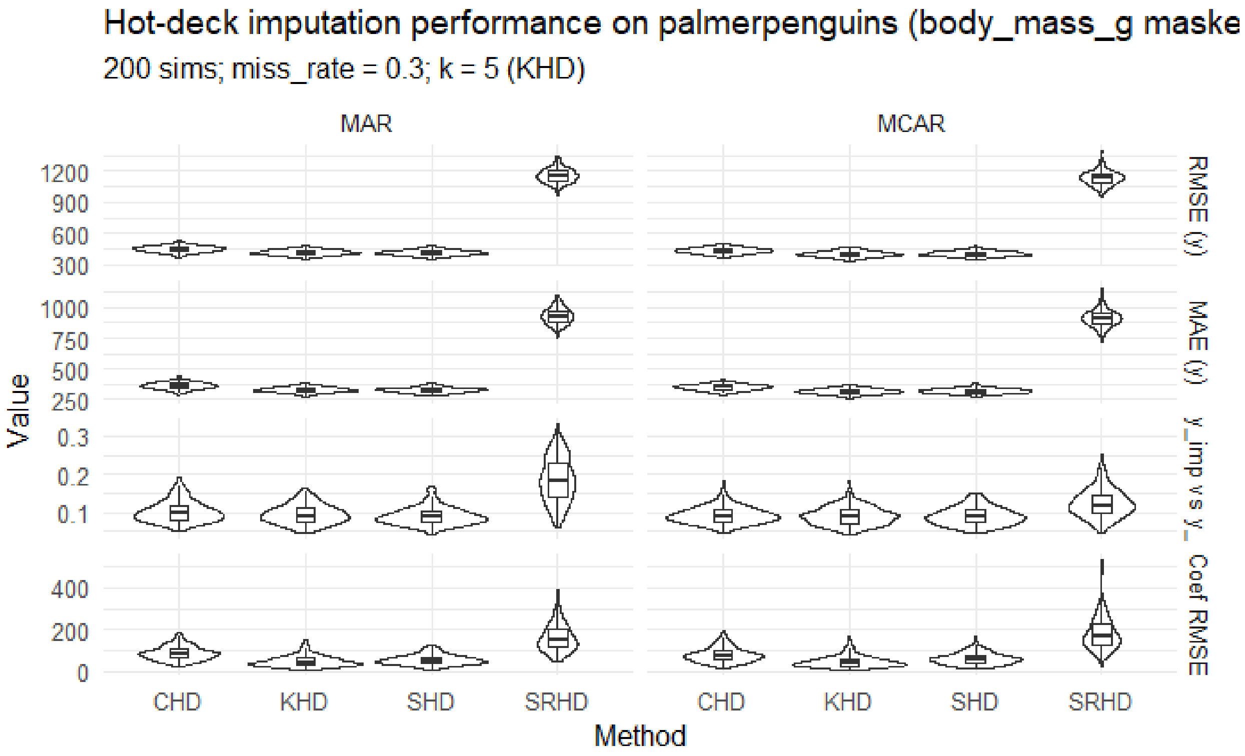
The simulation takes a few minutes to run and returns a lot of "no non-missing arguments to max" (or "min") warnings, but the code still runs to completion. We see that there are a lot of results to unpack; all of the methods seem to perform poorly (especially simple random hot-deck), but we need to recognize the scale of the variable of imputation which is body_mass_g (which ranges from 2700 to 6300 in the observed data). Thus, other than simple-random hot-deck, the performance of the hot-deck imputation methods is fairly decent.

It is unclear what is meant by the "gold" coefficients but I assume the last two columns of the table refer to differences in significant predictors for body_mass_g in the associated regression model.



	mech	method	RMSE_mean	RMSE_sd	MAE_mean	MAE_sd	KS_mean	KS_sd	CoefRMSE_mean	CoefRMSE_sd
1	MAR	SHD	414.2715	25.30142	327.7094	20.40371	0.09078285	0.02312549	56.11731	23.81642
2	MAR	KHD	416.4991	25.53909	330.1127	21.40414	0.09565658	0.02483801	49.91358	27.71490
3	MAR	CHD	450.4748	30.65026	362.2096	27.35113	0.10294890	0.02776790	87.68487	32.85648
4	MAR	SRHD	1155.8182	67.72962	931.6362	65.30131	0.18449928	0.05861900	161.12788	65.60793
5	MCAR	KHD	401.0713	28.28555	317.9888	22.86379	0.09131306	0.02359714	47.31560	27.64464
6	MCAR	SHD	404.7382	25.19337	320.8978	19.98143	0.09075609	0.02283646	62.16202	28.99140
7	MCAR	CHD	434.2948	27.96366	349.9232	24.10172	0.09332302	0.02176947	79.60359	34.51100
8	MCAR	SRHD	1131.3667	68.86077	911.6005	65.93667	0.12522260	0.03744046	181.46676	77.32722

The AI also provides a violin plot showing the distributions of various quantities across the simulations. They look fairly regular but there is not much more to say about them, albeit the KS statistic is instead listed as “y_imp vs y_” which can create some confusion if we don’t compare with the table. And, as we saw in the table, the plots for SRHD look vastly different from the others.



All in all, this simulation worked pretty well for its massive scale; interpretation is pretty light and broad (because the AI does not try to imagine what the results might be), but still we would be hard-pressed to get this much information from another source (or from simulating this ourselves) as quickly as we got it here.

Q: Why does the run_sims function return "Warning: no non-missing arguments to max; returning -Inf Warning: no non-missing arguments to min..." continuously as it's running?

Those warnings come from the **sequential hot-deck** (SHD) code when it tries to compute

```
left  <- max(obs_idx_ord[obs_idx_ord <  j], na.rm = TRUE)
right <- min(obs_idx_ord[obs_idx_ord >  j], na.rm = TRUE)
```

If there are **no observed donors to the left (or right)** of position `j` in the sorted order, the vectors inside `max()` / `min()` are **length 0**, so R emits:

> “no non-missing arguments to max/min; returning -Inf/Inf”

You then handle the `-Inf/Inf` with `is.infinite(...)`, so it *works*, but R still prints a warning **for every missing value at a boundary**, which is why you see them continuously.

Quick fix (no warnings)

Check lengths before calling `max/min`, and set `NA` (or branch) explicitly:

```
impute_shd <- function(df) {
  df2 <- df
  ord_by <- df2 %>%
    mutate(sex_num = as.integer(sex), sp_num = as.integer(species)) %>%
    transmute(sp_num, sex_num, flipper_length_mm,
bill_length_mm) %>%
    as.data.frame()

  o <- do.call(order, ord_by)
  inv_o <- integer(length(o)); inv_o[o] <- seq_along(o)

  y <- df2$body_mass_g[o]
  miss_idx_ord <- which(is.na(y))
  obs_idx_ord  <- which(!is.na(y))
  if (length(miss_idx_ord) == 0) return(df2)

  for (j in miss_idx_ord) {
    left_cands  <- obs_idx_ord[obs_idx_ord <  j]
    right_cands <- obs_idx_ord[obs_idx_ord >  j]

    left  <- if (length(left_cands))  max(left_cands)  else NA_integer_
    right <- if (length(right_cands)) min(right_cands) else NA_integer_

    donor <- if (is.na(left)) {
      right
    } else if (is.na(right)) {
      left
    } else {
      dL <- j - left; dR <- right - j
      if (dL < dR) left else if (dR < dL) right else sample(c(left, right), 1)
    }

    y[j] <- y[donor]
  }
  df2$body_mass_g[o] <- y
  df2
 }
```

Even cleaner (uses `findInterval`, no loops over donors)

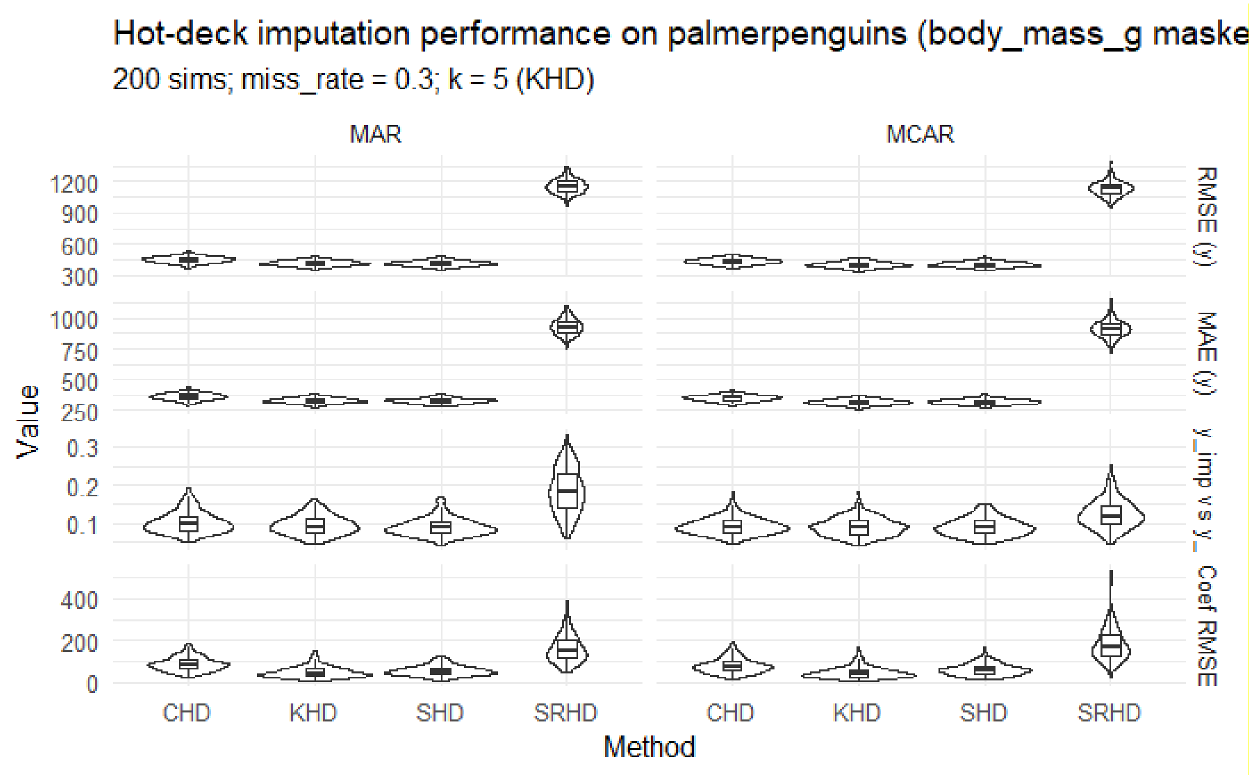
```
impute_shd <- function(df) {
 df2 <- df
 ord_by <- df2 %>%
  mutate(sex_num = as.integer(sex), sp_num = as.integer(species)) %>%
  transmute(sp_num, sex_num, flipper_length_mm, bill_length_mm) %>%
  as.data.frame()
 o <- do.call(order, ord_by)
 y <- df2$body_mass_g[o]
 miss <- is.na(y)
 obs  <- which(!is.na(y))
 if (!any(miss)) return(df2)

 # position between observed indices
 pos <- findInterval(which(miss), obs)
```

```
  left  <- ifelse(pos > 0,            obs[pos],      NA_integer_)
  right <- ifelse(pos < length(obs),   obs[pos + 1L],  NA_integer_)

  # choose nearest (break ties at random)
  choose_right <- (!is.na(right) & (is.na(left) | (right - which(miss) < which(miss) - left)))
  choose_left  <- (!is.na(left)  & (is.na(right) | (which(miss) - left < right - which(miss))))
  ties       <- (!is.na(left) & !is.na(right) & (right - which(miss) == which(miss) - left))

  donors <- integer(sum(miss))
  donors[choose_left]  <- left[choose_left]
  donors[choose_right] <- right[choose_right]
  if (any(ties)) donors[ties] <- ifelse(runif(sum(ties)) < 0.5, left[ties], right[ties])

  y[which(miss)] <- y[donors]
  df2$body_mass_g[o] <- y
  df2
}
```

Either version removes the boundary `max/min` calls on empty vectors, so the **warnings disappear** while keeping the SHD behavior intact.

Comments: The AI is able to correctly discern the source of the error, which might be hard to find by ourselves with how long the script is. It offers two suggestions for cleanliness that both successfully remove the warnings. Still, it's a bit strange that it bothered to offer two options, especially when it prefers the latter of the two.

Q: Interpret the results of the violin plot.



The violin plot is summarizing the **distribution of performance metrics** across 200 simulation runs, comparing four hot-deck methods (CHD, KHD, SHD, SRHD) under both **MAR** and **MCAR** missingness. Each row is a metric; each panel is a mechanism. Here's what the patterns show:

---

1. RMSE and MAE (predictive accuracy on imputed values)

- **SRHD (simple random)** performs **worst**: RMSE and MAE are much larger, with wide spread. This makes sense: SRHD ignores all auxiliary information, imputing from the marginal, so it misses systematic structure.

- **CHD, KHD, SHD** are **much lower and similar**, with small spread.
    - **KHD** (kNN using covariates) and **CHD** (species × sex) are slightly stronger because both exploit relevant predictors of body mass.
    - **SHD** (ordered matching) is competitive but a bit noisier.

---

2. KS statistic (distributional fidelity for imputed vs. true in masked rows)

- **SRHD** has the **largest KS**: its imputations don't align well with the conditional distribution of the missing cases.
- **CHD, KHD, SHD** show **much smaller KS**, so their imputed distributions are close to the true distribution.
    - **CHD** and **KHD** in particular preserve group-specific distributional shapes well.

---

3. Coefficient RMSE (fidelity of regression estimates)

- **SRHD** again inflates error—its imputations lead to regression coefficients further from the "gold" full-data values.
- **CHD, KHD, SHD** yield **much smaller coefficient RMSE**, so downstream modeling is more reliable.
- Differences between CHD, KHD, and SHD are modest; each captures predictor relationships enough to preserve regression estimates.

---

4. Comparison between MAR vs MCAR

- The **patterns are very consistent** across mechanisms: SRHD degrades sharply, while CHD/KHD/SHD cluster together with small errors.
- Under MAR, methods that exploit auxiliary information (CHD, KHD, SHD) shine even more, because missingness depends on covariates.
- Under MCAR, the advantage of smarter matching is smaller but still present.

✅ Overall interpretation:

- **SRHD (random draw)** is clearly inadequate: it preserves marginal distribution but loses predictive and modeling accuracy.
- **CHD, KHD, SHD** are **substantially better**, with **KHD and CHD** generally the most stable.
- Differences between MAR and MCAR aren't huge here, but MAR highlights the **advantage of conditioning on predictors**.

Would you like me to **rank the methods quantitatively** (e.g., average RMSE/Coef RMSE across sims) so we can say *which hot-deck dominates overall*?

Comments: The AI makes a few incorrect observations, expecting SRHD to "degrade sharply" from MCAR to MAR when actually the performance is about equally bad for both, and claiming that SHD is a bit noisier than KHD and CHD when actually CHD is the noisier of the three. This makes it seem like the AI was unable to properly analyze the violin plot and instead just fell back on its expectations from the relevant theory and context.

Q: Try again in an MNAR setting and see how the methods compare.

Great idea — MNAR is where donor–based methods (like hot-deck) often start to struggle, because **no auxiliary information in the observed data fully explains the missingness process**. Let's extend the simulation framework we used for MCAR and MAR to an **MNAR mechanism**.

Step 1. Define an MNAR mechanism

We'll make missingness in `body_mass_g` depend **directly on its (unobserved) value**:

$$P(R = 1 \mid Y, X) = \text{logit}^{-1}(\alpha_0 + \alpha_1 Y)$$

where:

- R=1 means "missing."
- Y = body mass.
- Larger (or smaller) body mass values have systematically higher missingness probability.

This violates MAR, because missingness depends on the unobserved Y itself.

We'll calibrate α_0 so that the overall missing rate is ~30%.

Step 2. Simulation code

Here's a modified extension to your earlier script:

```
# --- MNAR mask: depends directly on body_mass_g (unobserved value) ---
```

```
mask_mnar <- function(df, miss_rate = 0.30) {
  y <- df$body_mass_g
  # scale y so logit slope isn’t extreme
  y_std <- scale(y)
  lp_no_b0 <- -0.5 * y_std   # missingness increases with larger body mass
  # calibrate intercept
  f <- function(b0) mean(plogis(b0 + lp_no_b0)) - miss_rate
  b0 <- uniroot(f, interval = c(-20, 20))$root
  p  <- plogis(b0 + lp_no_b0)
  R  <- rbinom(length(y), 1, p) == 1
  df_imp <- df
  df_imp$body_mass_g[R] <- NA_real_
  list(data = df_imp, mask_idx = which(R))
}


# --- Extend one_run to include MNAR ---
one_run <- function(mech = c("MCAR","MAR","MNAR"), miss_rate = 0.30,
k = 5) {
  mech <- match.arg(mech)
  df_full <- dat0
  gold_fit <- lm(body_mass_g ~ bill_length_mm + bill_depth_mm +
flipper_length_mm + species + sex,
                 data = df_full)
  beta_gold <- coef(gold_fit)
  masked <- switch(mech,
                "MCAR" = mask_mcar(df_full, miss_rate),
                "MAR"  = mask_mar(df_full, miss_rate),
                "MNAR" = mask_mnar(df_full, miss_rate))
  df_miss  <- masked$data
  idx_miss <- masked$mask_idx

  # Impute
  imp_srhd <- impute_srhd(df_miss)
  imp_chd  <- impute_chd(df_miss)
  imp_khd  <- impute_khd(df_miss, k = k)
  imp_shd  <- impute_shd(df_miss)

  y_true <- df_full$body_mass_g[idx_miss]
  get_yimp <- function(df) df$body_mass_g[idx_miss]

  metrics <- function(df_imp, label) {
    y_imp <- get_yimp(df_imp)
    rmse  <- sqrt(mean((y_imp - y_true)^2))
    mae   <- mean(abs(y_imp - y_true))
    ks    <- ks_stat(y_imp, y_true)
    fit   <- lm(body_mass_g ~ bill_length_mm + bill_depth_mm + 
flipper_length_mm + species + sex,
                data = df_imp)
    beta  <- coef(fit)
    c(
      method = label,
      rmse = rmse,
      mae  = mae,
      ks   = ks,
      coef_rmse = coef_diff_rmse(beta, beta_gold, exclude_intercept = TRUE)
    )
  }

  bind_rows(
    as_tibble_row(metrics(imp_srhd, "SRHD")),
    as_tibble_row(metrics(imp_chd,  "CHD")),
    as_tibble_row(metrics(imp_khd,  "KHD")),
    as_tibble_row(metrics(imp_shd,  "SHD"))
  ) %>%
    mutate(mech = mech, miss_rate = miss_rate)
}

# --- Run sims for MNAR in addition to MCAR, MAR ---
res <- run_sims(n_sims = 200, miss_rate = 0.30, k = 5,
                mechs = c("MCAR","MAR","MNAR"))
```

```r
# Summarize
summary_tbl <- res %>%
  group_by(mech, method) %>%
  summarise(across(c(rmse, mae, ks, coef_rmse), list(mean=mean, sd=sd)), .groups="drop")

print(summary_tbl)

# Plot
p2 <- res %>%
  pivot_longer(c(rmse, mae, ks, coef_rmse),
               names_to = "metric", values_to = "value") %>%
  mutate(metric = factor(metric, levels = c("rmse","mae","ks","coef_rmse"),
                         labels = c("RMSE (y)","MAE (y)","KS (y_imp vs y_true)", "Coef RMSE"))) %>%
  ggplot(aes(x = method, y = value)) +
  geom_violin(trim = TRUE, alpha = 0.6) +
  geom_boxplot(width = 0.15, outlier.shape = NA) +
  facet_grid(metric ~ mech, scales = "free_y") +
  labs(title = "Hot-deck imputation performance on palmerpenguins",
       subtitle = "200 sims; miss_rate = 0.3; k = 5 (KHD)",
       x = "Method", y = "Value") +
  theme_minimal(base_size = 12)
```

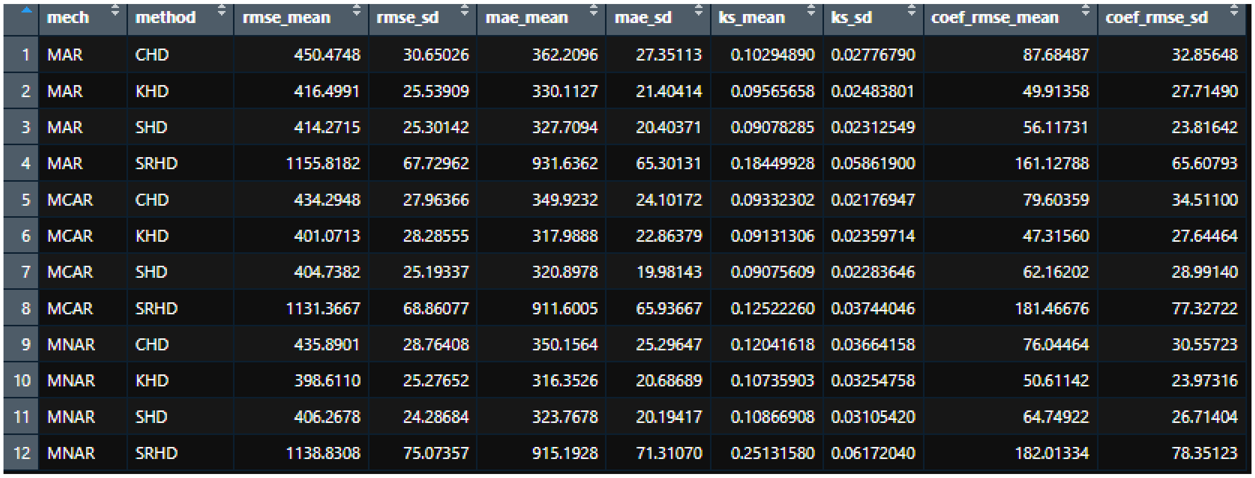
```
print(p2)
```

Step 3. What to expect in MNAR results

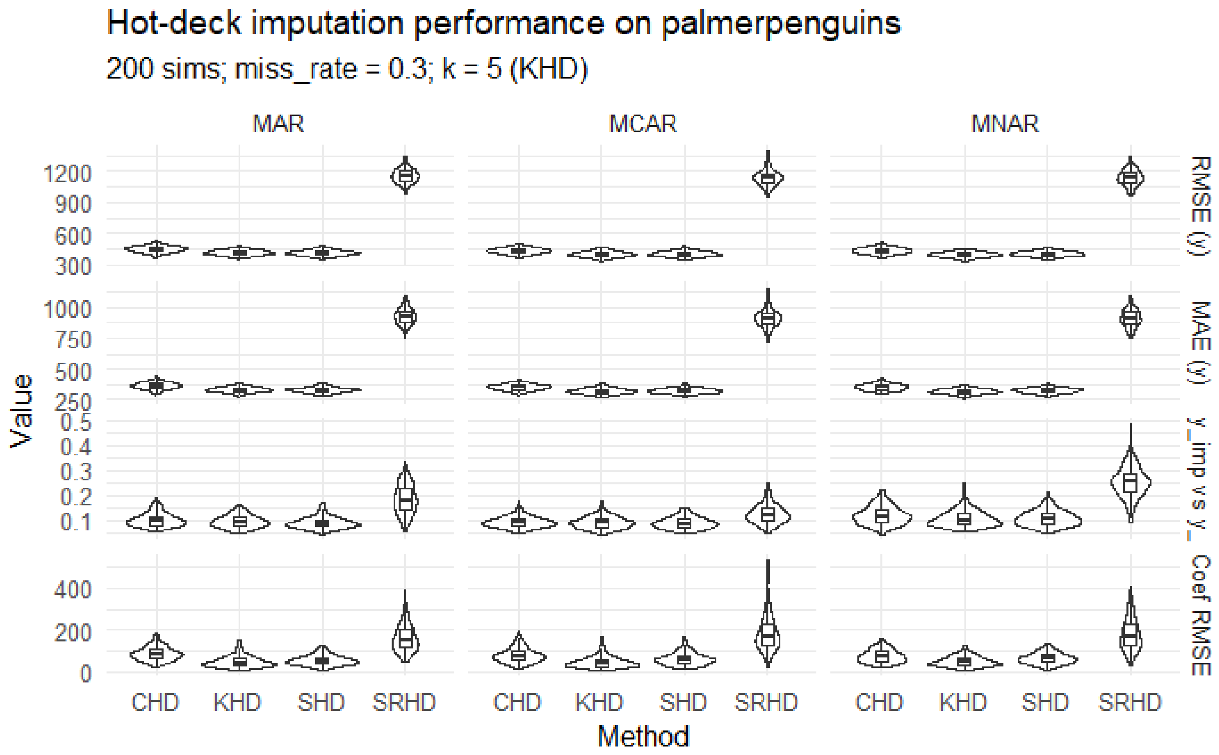
- **All methods degrade** because missingness is linked to `body_mass_g` itself, which no auxiliary can fully capture.
- **Bias creeps in**: even CHD and KHD will tend to underestimate or overestimate mean body mass, depending on which end of the distribution is missing more often.
- **SRHD** remains worst, but the **gap between SRHD and smarter methods shrinks**, since even CHD/KHD/SHD cannot fully recover the truth.
- **Coefficient RMSE**: model estimates become biased, especially for the intercept and for predictors correlated with body mass.
- **KS statistic**: larger discrepancies appear, because imputed values can't match the distribution of truly missing values (which are systematically skewed).

👉 Would you like me to **run the code and show you the MNAR violin plot and summary table**, or just leave the script for you to execute in your own R environment?

Comments: The AI creates a short extension to the existing code that reimplements the methods in an MNAR setting. The code runs smoothly and returns the following results:



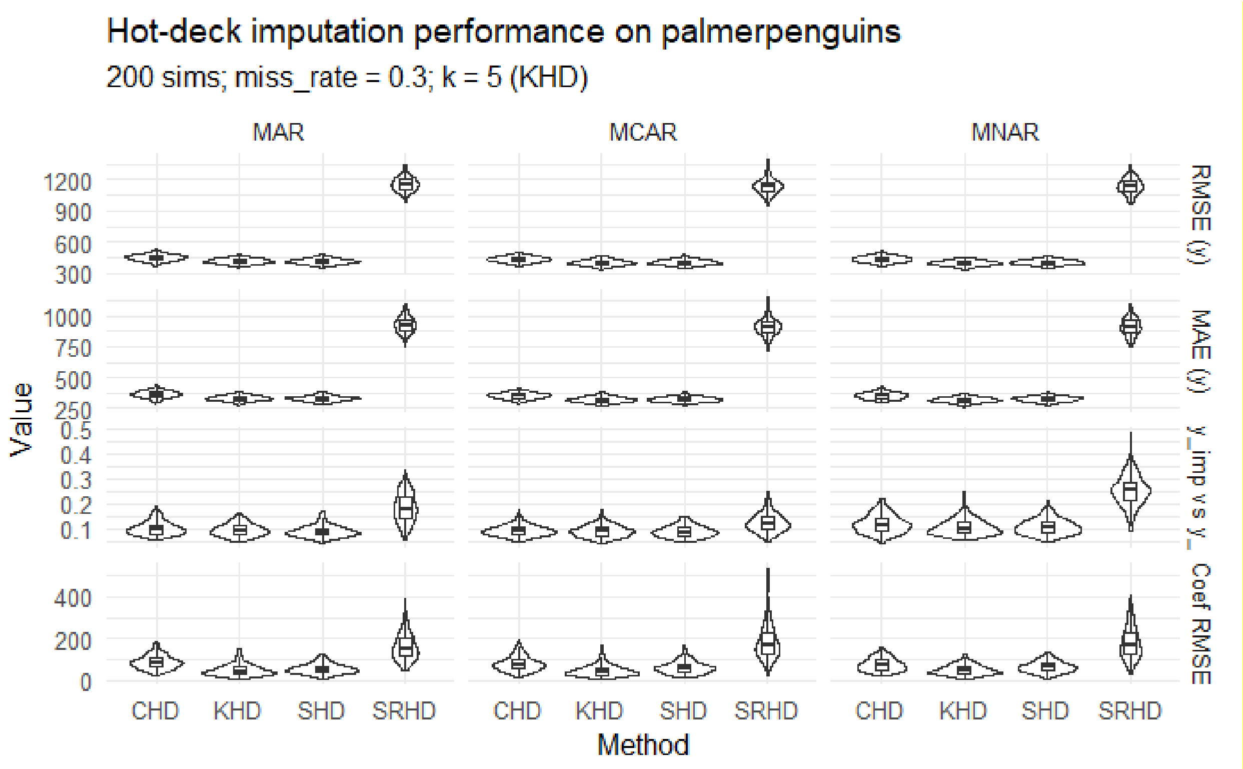
	mech	method	rmse_mean	rmse_sd	mae_mean	mae_sd	ks_mean	ks_sd	coef_rmse_mean	coef_rmse_sd
1	MAR	CHD	450.4748	30.65026	362.2096	27.35113	0.10294890	0.02776790	87.68487	32.85648
2	MAR	KHD	416.4991	25.53909	330.1127	21.40414	0.09565658	0.02483801	49.91358	27.71490
3	MAR	SHD	414.2715	25.30142	327.7094	20.40371	0.09078285	0.02312549	56.11731	23.81642
4	MAR	SRHD	1155.8182	67.72962	931.6362	65.30131	0.18449928	0.05861900	161.12788	65.60793
5	MCAR	CHD	434.2948	27.96366	349.9232	24.10172	0.09332302	0.02176947	79.60359	34.51100
6	MCAR	KHD	401.0713	28.28555	317.9888	22.86379	0.09131306	0.02359714	47.31560	27.64464
7	MCAR	SHD	404.7382	25.19337	320.8978	19.98143	0.09075609	0.02283646	62.16202	28.99140
8	MCAR	SRHD	1131.3667	68.86077	911.6005	65.93667	0.12522260	0.03744046	181.46676	77.32722
9	MNAR	CHD	435.8901	28.76408	350.1564	25.29647	0.12041618	0.03664158	76.04464	30.55723
10	MNAR	KHD	398.6110	25.27652	316.3526	20.68689	0.10735903	0.03254758	50.61142	23.97316
11	MNAR	SHD	406.2678	24.28684	323.7678	20.19417	0.10866908	0.03105420	64.74922	26.71404
12	MNAR	SRHD	1138.8308	75.07357	915.1928	71.31070	0.25131580	0.06172040	182.01334	78.35123



We don't actually see much different between the methods in the MNAR setting versus the other two; if anything, the KS statistics are worse, but not by much (except in the case of SRHD). It is likely that the missingness depending on the response was not enough to obscure the X-Y relationships completely, or at least no more than in the MAR case. This contradicts the AI's expectations for a notable worsening of the methods; it does not anticipate that changing the missingness mechanism might not actually alter the observed data substantially (for example).

Notably, R = 1 is defined as “missing” here, whereas in many past examples it was defined as “observed.” This shows a general inconsistency in the AI’s definition of the variable that requires us to pay careful attention to how it’s constructed in the situation at hand.

Q: Why might the MNAR scenario not be substantially worse for these methods, as we see here?



That’s a really sharp observation 👀 — in your plot, the **MNAR panel doesn’t look much worse than MAR/MCAR**, even though in theory MNAR should cause problems. Here are the main reasons why your MNAR simulation didn’t degrade hot-deck performance very much:

---

1. Auxiliary variables are highly correlated with body mass

- In penguins, `bill_length_mm`, `bill_depth_mm`, `flipper_length_mm`, `species`, and `sex` are all **strong predictors of body mass**.
- Even though missingness depends directly on body mass (MNAR), the auxiliaries are so correlated that **conditioning on them almost reconstructs body mass**.

- In other words, the MAR approximation is “nearly correct” in this dataset.

---

2. Moderate missingness fraction

- At **30% missing**, the donor pool remains large, so all methods (especially CHD and KHD) have plenty of good matches.

- Severe MNAR effects often show up more clearly at **higher missingness levels (50–70%)**, when biased selection cuts deeper into the tails of the distribution.

---

3. MNAR mechanism was fairly mild

- The logit slope in your `mask_mnar` function was set to about `-0.5 * scaled(Y)`.

- That induces some dependence of missingness on `Y`, but not a strong one — missingness isn’t extremely concentrated in the heaviest birds.

- So the **departure from MAR is modest**, and the effect looks similar to MAR in practice.

---

4. Evaluation metrics focus on masked rows

- RMSE, MAE, KS are computed on the **true vs imputed values for the missing cases**.

- Because hot-deck imputes from similar donors, it can still approximate those missing values well, even if the missingness was MNAR.

- What you're not measuring here is the **bias in full-sample estimates** (means, variances, regression intercepts). Those would usually show more distortion under MNAR.

---

5. Donor-based methods naturally preserve distributions

- Unlike model-based methods that might extrapolate incorrectly, hot-deck draws real observed values.

- This makes them somewhat robust, because imputations will always resemble observed data, even when MNAR is present.

✅ **Summary:**
The MNAR you introduced is fairly mild and highly "masked" by strong auxiliary predictors. Hot-deck methods, especially CHD and KHD, can still find close donors, so performance looks similar across MAR and MNAR. To really expose MNAR's harm, you'd need either:

- Stronger MNAR dependence on `Y`,

- Weaker auxiliaries (less correlation with `Y`),

- Higher missingness fractions,

- Or evaluation of population-level biases (e.g., mean body mass, regression intercept).

Would you like me to **redesign the MNAR mechanism** (e.g., very steep slope so nearly all heavy birds are missing) and re-run the simulation to show a clearer breakdown in hot-deck performance?

Comments: When pressed on the similar performance between MAR and MNAR, the AI is able to identify several good reasons why there is little discrepancy between results here. This answer was easy to follow and made several good points; I mostly just wish some of this discussion was included in the previous response.

Q: Try a more harsh MNAR mechanism and recompare.

You're right—let's make MNAR truly punishing and re-run the same comparison. Two stronger mechanisms:

1. **Residual-MNAR (hard):** Missingness depends on the *residual* of body mass after conditioning on auxiliaries, so no combination of X's can "explain away" the missingness:

$$P(R{=}1\mid Y,X)=\text{logit}^{-1}\big(b_0+\gamma\,\underbrace{\text{scale}\{Y-\hat{\mathbb E}(Y\mid X)\}}_{\text{conditional residual}}\big)$$,

with a **steep** slope (e.g., $\gamma=-4.5$) so high-residual birds go missing.

2. **Tail-censor MNAR (step):** Roughly remove the **top 50%** of body masses with high probability (0.95) and keep the bottom with low probability (0.05)—this removes donors exactly where you need them.

Drop these into your script (keep your fixed `impute_shd`):

```
# --- Harsher MNAR mechanisms ---


# (A) Residual-MNAR: depends on Y residual after regressing on auxiliaries (X)

mask_mnar_resid_strong <- function(df, miss_rate = 0.50, slope = -4.5) {

 form <- body_mass_g ~ bill_length_mm + bill_depth_mm + flipper_length_mm + species + sex

 fit  <- lm(form, data = df)

 e    <- resid(fit)

 e_z  <- as.numeric(scale(e))
```

```
  f <- function(b0) mean(plogis(b0 + slope * e_z)) - miss_rate
  b0 <- uniroot(f, c(-20, 20))$root
  p  <- plogis(b0 + slope * e_z)
  R  <- rbinom(nrow(df), 1, p) == 1
  out <- df; out$body_mass_g[R] <- NA_real_
  list(data = out, mask_idx = which(R))
}
```

```
# (B) Tail-censor MNAR (step function around a quantile threshold)
mask_mnar_tail_step <- function(df, miss_rate = 0.50, p_hi = 0.95, p_lo = 0.05) {
  y_z <- as.numeric(scale(df$body_mass_g))
    # choose threshold so expected missing ≈ miss_rate
  g <- function(q) mean(ifelse(y_z > q, p_hi, p_lo)) - miss_rate
  qstar <- uniroot(g, c(min(y_z)-1, max(y_z)+1))$root
  p <- ifelse(y_z > qstar, p_hi, p_lo)
  R <- rbinom(nrow(df), 1, p) == 1
  out <- df; out$body_mass_g[R] <- NA_real_
  list(data = out, mask_idx = which(R))
}
```

Wire these into your runner:

```
one_run <- function(mech = c("MCAR","MAR","MNAR_resid","MNAR_tail"),
              miss_rate = 0.50, k = 5) {
  mech <- match.arg(mech)
  df_full <- dat0
  gold_fit <- lm(body_mass_g ~ bill_length_mm + bill_depth_mm +
flipper_length_mm + species + sex,
             data = df_full)
  beta_gold <- coef(gold_fit)

  masked <- switch(mech,
    "MCAR"       = mask_mcar(df_full, miss_rate),
    "MAR"        = mask_mar(df_full, miss_rate),
    "MNAR_resid" = mask_mnar_resid_strong(df_full, miss_rate),
    "MNAR_tail"  = mask_mnar_tail_step(df_full, miss_rate)
  )
  df_miss  <- masked$data
  idx_miss <- masked$mask_idx

  imp_srhd <- impute_srhd(df_miss)
  imp_chd  <- impute_chd(df_miss)
  imp_khd  <- impute_khd(df_miss, k = k)
  imp_shd  <- impute_shd(df_miss)

  y_true <- df_full$body_mass_g[idx_miss]
  get_yimp <- function(df) df$body_mass_g[idx_miss]
  metrics <- function(df_imp, label) {
    y_imp <- get_yimp(df_imp)
    rmse  <- sqrt(mean((y_imp - y_true)^2))
    mae   <- mean(abs(y_imp - y_true))
    ks    <- ks_stat(y_imp, y_true)
    fit   <- lm(body_mass_g ~ bill_length_mm + bill_depth_mm + flipper_length_mm + species + sex,
                data = df_imp)
    beta  <- coef(fit)
    mean_bias <- mean(df_imp$body_mass_g) - mean(df_full$body_mass_g)   # expose MNAR bias
    int_bias  <- beta["(Intercept)"] - beta_gold["(Intercept)"]
    c(method = label, rmse = rmse, mae = mae, ks = ks,
      coef_rmse = coef_diff_rmse(beta, beta_gold, exclude_intercept = TRUE),
      mean_bias = mean_bias, intercept_bias = int_bias)
  }

  dplyr::bind_rows(
    tibble::as_tibble_row(metrics(imp_srhd, "SRHD")),
    tibble::as_tibble_row(metrics(imp_chd,  "CHD")),
    tibble::as_tibble_row(metrics(imp_khd,  "KHD")),
    tibble::as_tibble_row(metrics(imp_shd,  "SHD"))
  ) %>% mutate(mech = mech, miss_rate = miss_rate)
```

```
}

# Re-run with harsher scenarios and higher missingness to stress methods
res <- run_sims(n_sims = 200, miss_rate = 0.50, k = 5,
                mechs = c("MCAR","MAR","MNAR_resid","MNAR_tail"))

# Summaries & plot (add the new metrics too)
summary_tbl <- res %>%
  group_by(mech, method) %>%
  summarise(across(c(rmse, mae, ks, coef_rmse, mean_bias, intercept_bias),
                   list(mean = mean, sd = sd)), .groups = "drop")
print(summary_tbl)

p <- res %>%
  pivot_longer(c(rmse, mae, ks, coef_rmse, mean_bias, intercept_bias),
               names_to = "metric", values_to = "value") %>%
  mutate(metric = factor(metric, levels =
c("rmse","mae","ks","coef_rmse","mean_bias","intercept_bias"),
                         labels = c("RMSE (y)","MAE (y)","KS (y_imp vs y_true)",
                                    "Coef RMSE","Mean Bias","Intercept Bias"))) %>%
  ggplot(aes(x = method, y = value)) +
  geom_violin(trim = TRUE, alpha = 0.6) +
  geom_boxplot(width = 0.15, outlier.shape = NA) +
  facet_grid(metric ~ mech, scales = "free_y") +
  labs(title = "Hot-deck Imputation under Harsher MNAR",
       subtitle = "200 sims; miss_rate = 0.5; residual- and tail-based MNAR",
       x = "Method", y = "Value") +
  theme_minimal(base_size = 12)
print(p)
```

What you should see

- **Both MNAR panels** deteriorate vs MAR/MCAR.
- **Residual-MNAR:** biggest hit to **Coef RMSE**, **mean bias**, and **intercept bias**—even CHD/KHD can't fix what's in the conditional residual.
- **Tail-MNAR:** sharp rise in **RMSE/MAE/KS** because donors for the upper tail are scarce; hot-deck pulls from the middle, shrinking extremes.
- **SRHD** remains worst; `KHD ≳ CHD ≳ SHD` typically, but all degrade notably in MNAR—especially on **Mean/Intercept bias**.

If you want, we can also add **Rubin's MI** with multiple hot-deck draws and show how between-imputation variance reacts under these harsher MNARs.

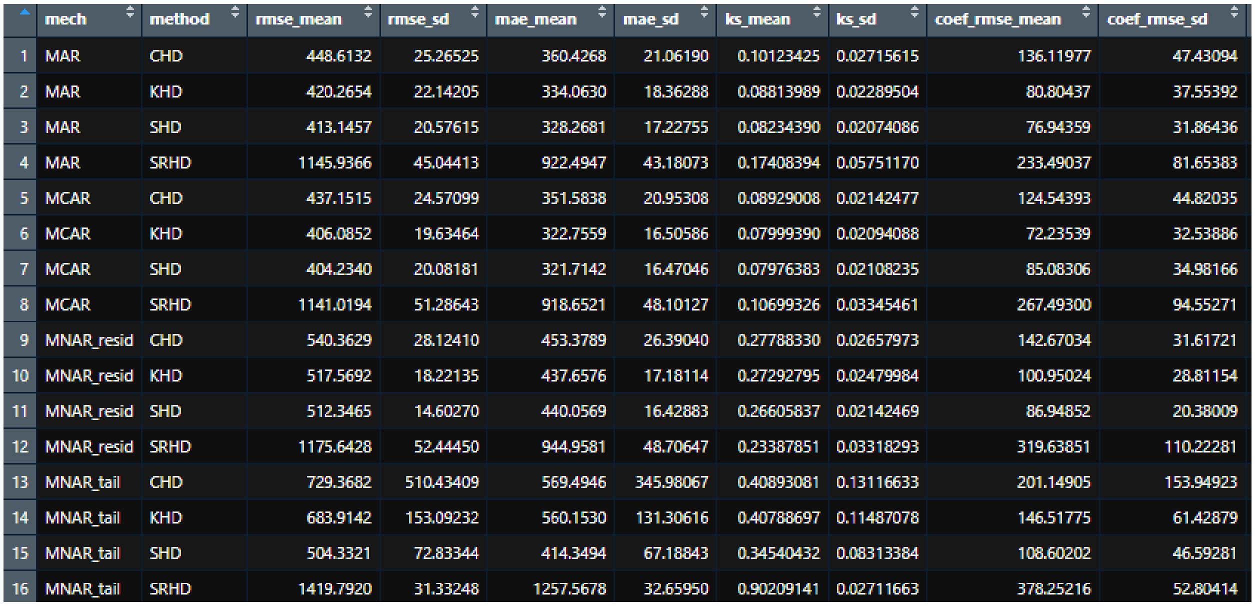
Comments: There are some small issues keeping the code from running properly: First, the "intercept_bias" term is for some reason being named in the "res" data frame as "intercept_bias.(Intercept)." This is fixed by using

```
        names(res)[names(res) == 'intercept_bias.(Intercept)'] <- 'intercept_bias'
```

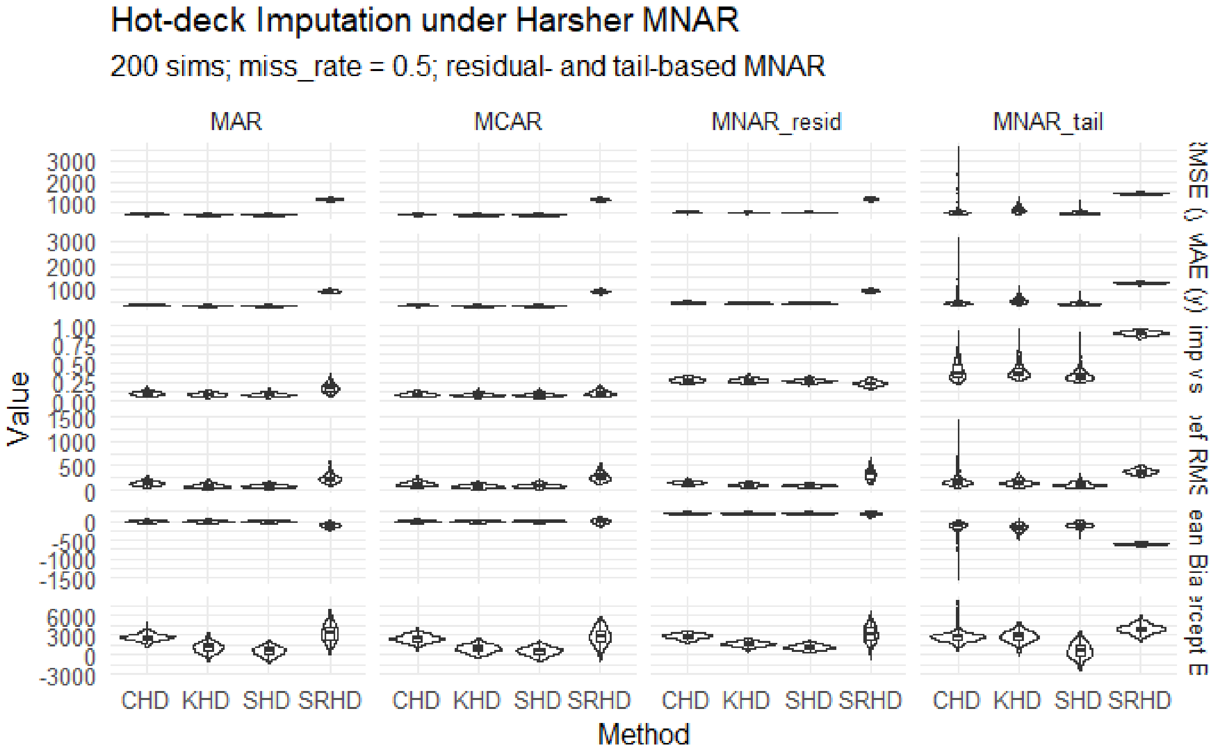
Then for some reason the "mean_bias" and "intercept_bias" terms are treated as characters, so we need to also employ the fix

```
        res$mean_bias <- as.numeric(res$mean_bias)

        res$intercept_bias <- as.numeric(res$intercept_bias)
```

Doing this allows the code to run properly, and we see the following results:



	mech	method	rmse_mean	rmse_sd	mae_mean	mae_sd	ks_mean	ks_sd	coef_rmse_mean	coef_rmse_sd
1	MAR	CHD	448.6132	25.26525	360.4268	21.06190	0.10123425	0.02715615	136.11977	47.43094
2	MAR	KHD	420.2654	22.14205	334.0630	18.36288	0.08813989	0.02289504	80.80437	37.55392
3	MAR	SHD	413.1457	20.57615	328.2681	17.22755	0.08234390	0.02074086	76.94359	31.86436
4	MAR	SRHD	1145.9366	45.04413	922.4947	43.18073	0.17408394	0.05751170	233.49037	81.65383
5	MCAR	CHD	437.1515	24.57099	351.5838	20.95308	0.08929008	0.02142477	124.54393	44.82035
6	MCAR	KHD	406.0852	19.63464	322.7559	16.50586	0.07999390	0.02094088	72.23539	32.53886
7	MCAR	SHD	404.2340	20.08181	321.7142	16.47046	0.07976383	0.02108235	85.08306	34.98166
8	MCAR	SRHD	1141.0194	51.28643	918.6521	48.10127	0.10699326	0.03345461	267.49300	94.55271
9	MNAR_resid	CHD	540.3629	28.12410	453.3789	26.39040	0.27788330	0.02657973	142.67034	31.61721
10	MNAR_resid	KHD	517.5692	18.22135	437.6576	17.18114	0.27292795	0.02479984	100.95024	28.81154
11	MNAR_resid	SHD	512.3465	14.60270	440.0569	16.42883	0.26605837	0.02142469	86.94852	20.38009
12	MNAR_resid	SRHD	1175.6428	52.44450	944.9581	48.70647	0.23387851	0.03318293	319.63851	110.22281
13	MNAR_tail	CHD	729.3682	510.43409	569.4946	345.98067	0.40893081	0.13116633	201.14905	153.94923
14	MNAR_tail	KHD	683.9142	153.09232	560.1530	131.30616	0.40788697	0.11487078	146.51775	61.42879
15	MNAR_tail	SHD	504.3321	72.83344	414.3494	67.18843	0.34540432	0.08313384	108.60202	46.59281
16	MNAR_tail	SRHD	1419.7920	31.33248	1257.5678	32.65950	0.90209141	0.02711663	378.25216	52.80414



I suspect that the design of the residual-based MNAR mechanism was not as robust as the AI expected, because despite about 50% missingness we do not see substantial deviation from the MAR/MCAR mechanisms. It makes higher residual birds more likely to be missing, but these birds were already a poor fit to the predictors so perhaps they were not modeled well in the other scenarios either.

The auxiliary predictors are not the only reason for good performance from these methods in MNAR that the AI identified, so it could be that other factors are at play. However, when the MNAR mechanism removes virtually all of the heavier birds from the observed sample, the MAR methods break down substantially, and imputations become extremely poor. This suggests that these methods strongly rely on fit information obtained from these heavier birds.

The AI doesn't especially expect a difference between the two MNAR scenarios; it thinks both of them should degrade performance. While this is a reasonable expectation, we see a stark disparity between the poor performance from tail-censoring MNAR and the modest performance from residual-based MNAR. It is true, however, that SRHD remains the worst method in every scenario.

Tab 4

Q: When will MAR methods perform well under MNAR missingness?

Short answer: MAR-based methods (MI, FIML, likelihood under ignorability) can work surprisingly well even when the *truth* is MNAR if the part of the data-generating process that violates MAR either (i) is small, (ii) cancels in the estimand you care about, or (iii) only shifts aspects of the distribution that your estimator is orthogonal to. Here are the main cases worth knowing.

When MAR works under MNAR

1. **“Near-MAR” or small departures.**
   Write a pattern–mixture decomposition

$$E(Y\#\#, R=1) = m(X).E(Y \mid X, R=0) = m(X) + \Delta(X), \qquad E(Y \mid X, R=1) = m(X).$$

Imputing under MAR sets $\Delta(\cdot) \equiv 0$. For the marginal mean,

$$\mathrm{Bias}(\hat{\mu}_{\mathrm{MAR}}) = P(R=0)\, E[\Delta(X) \mid R=0].$$

So if either the missing fraction P(R=0) is small or the average departure $E[\Delta(X) \mid R=0]$ is small (because Δ(X) is weak or changes sign and averages out), MAR will look good. Same logic applies to many smooth functionals.

2. **Rich X that “almost explains” missingness.**
   If you include strong predictors of Y and of R in the imputation/analysis model, the remaining dependence of R on Y *given X* (i.e., Δ(X)) is often small. In practice, lots of good auxiliaries make MNAR feel like “MAR plus a tiny residual.”

3. **Residual-only selection ⇒ slopes OK.**
   Linear model $Y = X^\top \beta + \varepsilon$ with $E(\varepsilon \mid X) = 0$. Suppose missingness depends on Y only through the residual: $R \not\perp Y \mid X$ but $R \perp X \mid \varepsilon$ (MNAR). Then

$$\mathrm{Cov}(X, \varepsilon \mid R=1) = 0$$

because X is independent of ε and of R. OLS/FIML/MI fitted under MAR recover β consistently; only the **intercept** is biased (a location shift). So slope estimates (and contrasts that depend only on slopes) can be very robust to this MNAR form.

4. **Constant shift MNAR cancels in contrasts.**
   If the MNAR departure is an X-invariant offset, $E(Y \mid X, R=0) = m(X) + \delta$, then any estimand that differences out the intercept—e.g., treatment–control mean *difference* with the same δ in both arms, or regression slopes—remains (nearly) unbiased even though the marginal mean is biased.

5. **Strong overlap / no tail MNAR.**
   If missing cases live in regions of X-space that are well represented among observed cases, MAR imputation avoids hard extrapolation. Even with MNAR, bias stays small when the MNAR mechanism mostly perturbs the *tails* and you aren't forced to impute far beyond observed support.

6. **Low missingness.**
   For many smooth parameters, first-order bias scales like "missing fraction × size of MNAR departure." With, say, 5–10% missingness, even moderate MNAR often produces tiny bias relative to sampling error.

Practical signs you're in a "safe" regime

- **High R^2 for** $Y \mid X$ and **weak association of R** with the **residuals** from a good $Y \mid X$ model (regress R on $\hat{\varepsilon}$; a flat relationship suggests near-MAR or residual-only MNAR where slopes are safe).

- **Auxiliary variables** that predict both Y and missingness are in the imputation model.

- **Sensitivity/tipping-point** checks (pattern-mixture with +δ to the imputed values): if you need implausibly large δ to move conclusions, MAR is effectively fine.

- **Agreement across methods** (e.g., MI vs. flexible IPW vs. complete-case slopes) for the estimand of interest.

Takeaways by estimand

- **Marginal means / prevalence:** sensitive; MAR works when p_{miss} and average Δ are small.

- **Regression slopes / treatment effects (differences):** often robust if MNAR is a constant shift or residual-only selection; intercepts can still be off.

- **Interactions / nonlinear functionals:** more fragile—require that $\Delta(X)$ not interact with X.

If you want, I can spin up a quick simulation showing: (i) residual-only MNAR where slopes are recovered under MAR, versus (ii) outcome-level MNAR where both intercept and slopes drift—so you can see the contrast numerically.

Comments: This response is fairly high-level for an initial response at the start of a discussion, and I wasn't able to fully follow the setup because it assumes the user is already familiar with pattern-mixture decomposition. Still, there are some concepts (especially those explained verbally) that are easier to understand and useful considerations when imputing under MNAR (we have observed some of them in previous simulations).

Q: Can you explain the pattern-mixture decomposition and bias computation more clearly?

Great question—let's make the pattern–mixture view and the bias formula precise, step by step. I'll keep X fully observed to focus the ideas on Y and the missingness indicator $R \in \{0,1\}$ (1 = observed, 0 = missing).

1) Pattern–mixture factorization

For each covariate value X=x, factor the joint law of (Y,R) as

$$p(y, r \mid x) = p(r \mid x)\, p(y \mid x, r).$$

Define

$$\pi(x) := P(R = 0 \mid X = x), \qquad m_r(x) := E(Y \mid X = x, R = r) \quad (r \in \{0,1\}).$$

Then by the law of total expectation,

$$E(Y \mid X = x) = (1 - \pi(x))\, m_1(x) + \pi(x)\, m_0(x).$$

The **pattern–mixture departure** from MAR is

$$\Delta(x) := m_0(x) - m_1(x).$$

Under MAR (ignorable), $p(y \mid x, R = 0) = p(y \mid x, R = 1)$ so $\Delta(x) \equiv 0$. Under MNAR, Δ(x) encodes how the missing cases differ from the observed cases *at the same x.*

2) The marginal mean and its MAR-imputed limit

Suppose we want the **population mean** $\mu := E(Y)$. Using the identity above,

$$\mu = E_X\big[(1 - \pi(X))\, m_1(X) + \pi(X)\, m_0(X)\big] = E\big[m_1(X)\big] + E\big[\pi(X)\, \Delta(X)\big]. \qquad (\#\#)$$

What does a **MAR-based imputation** (or likelihood under ignorability) target? As $n \to \infty$, if you fit a correct model for $m_1(x) = E(Y \mid X = x, R = 1)$ and then impute missing Y's **as if** $m_0(x) = m_1(x)$ (i.e., you set $\Delta \equiv 0$), your estimator's probability limit is

$$\mu_{\mathrm{MAR}} = E\big[m_1(X)\big].$$

Subtracting from the truth (★) gives the bias:

$$\boxed{\mathrm{Bias}(\hat{\mu}_{\mathrm{MAR}}) = E[\hat{\mu}_{\mathrm{MAR}}] - \mu = -\, E\big[\pi(X)\,\Delta(X)\big].}$$

Sign intuition. If the missing cases have ***larger*** means than the observed at the same x (so Δ(x)>0), MAR-imputation ***underestimates*** μ (negative bias), because it fills in with $m_1(x)$ instead of the larger $m_0(x)$.

Two useful special cases:

- **Constant shift MNAR:** $\Delta(x) \equiv \delta$. Then B$\mathrm{Bias}(\hat{\mu}_{\mathrm{MAR}}) = -\,\delta\, P(R = 0)$.

  Small missingness or a tiny δ ⇒ tiny bias.

- **Averaging out within the missing stratum:** Using $E[\pi(X)\Delta(X)] = P(R = 0)\, E[\Delta(X) \mid R = 0]$, if $E[\Delta(X) \mid R = 0] = 0$ (departures change sign and cancel on average among the missing), the MAR estimator of μ\mu is unbiased despite MNAR.

3) Conditional targets and “local” bias

For the **conditional mean at a specific** x_0,

$$\theta(x_0) := E(Y \mid X = x_0) = (1 - \pi(x_0))m_1(x_0) + \pi(x_0)m_0(x_0),$$

the MAR plug-in $\theta_{\mathrm{MAR}}(x_0) = m_1(x_0)$ has bias

$$\boxed{\mathrm{Bias}\big(\hat{\theta}_{\mathrm{MAR}}(x_0)\big) = -\,\pi(x_0)\,\Delta(x_0).}$$

So even if the marginal mean is nearly unbiased (because Δ averages to ~0), there can be **heterogeneous** local bias where π(x) or Δ(x) is large.

4) Why slopes can be robust under some MNAR

Consider the linear model $Y = X^\top \beta + \varepsilon$ with $E(\varepsilon \mid X) = 0$, and suppose missingness depends on the **residual** but not on X: $P(R = 1 \mid X, \varepsilon) = g(\varepsilon)$. Then

$$m_1(x) = E(Y \mid X = x, R = 1) = x^\top \beta + E(\varepsilon \mid R = 1), \quad m_0(x) = x^\top \beta + E(\varepsilon \mid R = 0),$$

so $\Delta(x) \equiv E(\varepsilon \mid R = 0) - E(\varepsilon \mid R = 1) =: \delta$ is a **constant shift**. In the observed data,

$$\mathrm{Cov}(X, \varepsilon \mid R = 1) = 0$$

(because selection depends only on ε, not on X, leaving X and ε uncorrelated within R=1). Hence OLS (and correctly specified MAR-based likelihood/MI) still recover β consistently—**slopes are fine**—while the intercept is shifted by $E(\varepsilon \mid R = 1)$. For the marginal mean, the bias formula above simplifies to $-\delta\, P(R = 0)$.

5) How to use the formula in practice

- To **bound bias** for μ, posit a plausible Δ(x) (pattern–mixture sensitivity): e.g., Δ(x) = δ or $\Delta(x) = \gamma^\top s(x)$, and compute $-E[\pi(X)\Delta(X)]$ using your data's $\hat{\pi}(X)$ and X-distribution.
- For **treatment differences**, compute the bias **within arm** and subtract: $\mathrm{Bias}(\hat{\tau}_{\mathrm{MAR}}) = -E[\pi(X)\Delta(X) \mid A{=}1] + E[\pi(X)\Delta(X) \mid A{=}0]$.

  If Δ and π are (approximately) the same across arms, the biases cancel and $\hat{\tau}$ is robust.

Bottom line. The pattern–mixture decomposition cleanly isolates **where** MNAR lives via Δ(x). MAR-based estimators behave well when the weighted average $E[\pi(X)\Delta(X)]$ is small (small missingness, small/oscillating departures, or cancellations in your estimand), and many regression **slopes** are robust when MNAR acts like an X-invariant shift.

Comments: The AI is a little too "talkative" and almost overwhelms with information, but for the most part I think this does add good clarity to the original response. The pattern-mixture factorization is made much more clear by this response and notation is used effectively. I was able to understand the first response much better through this one, even if there's too much information to fully understand everything.

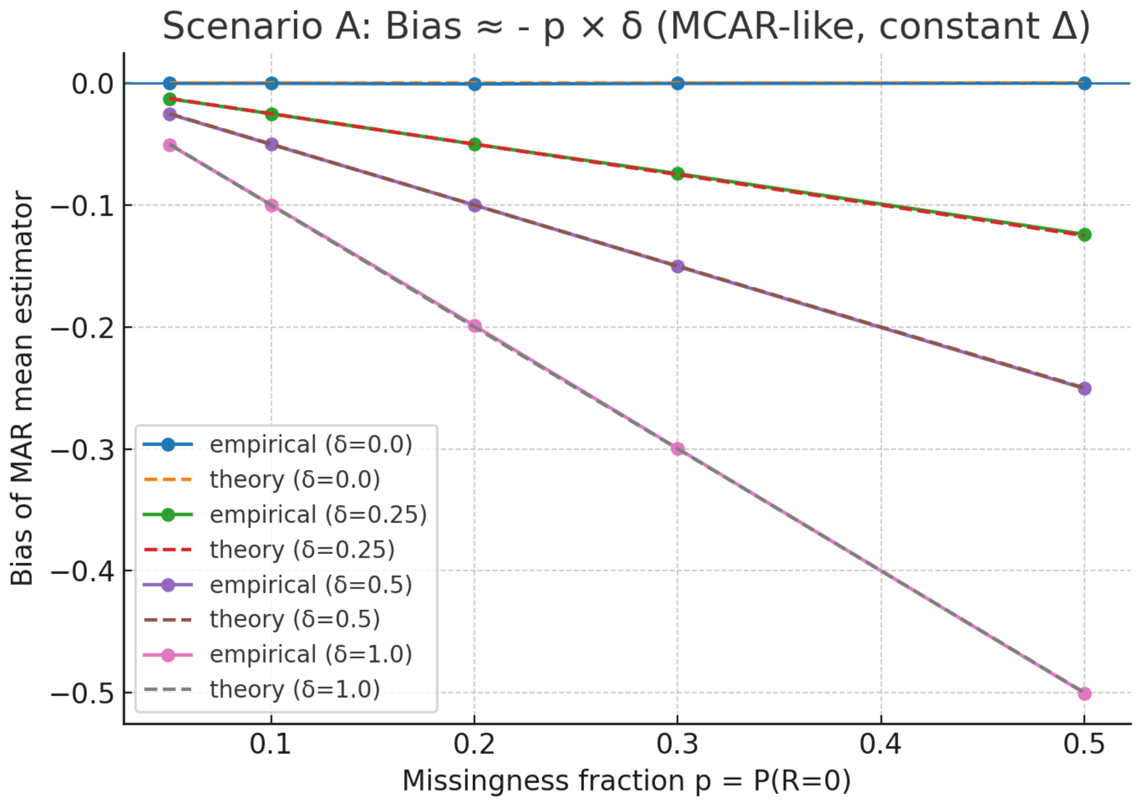
Q: Can you create a simple simulated demo to show the bias relationship to missingness and delta(X)?

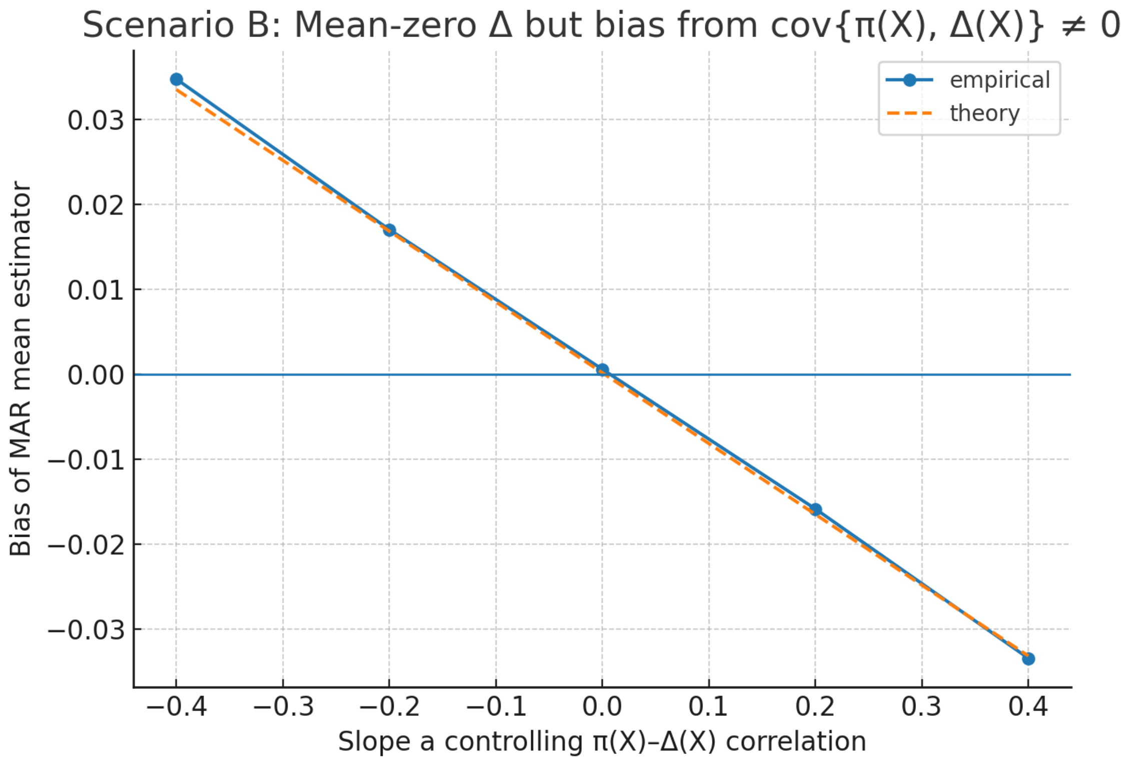
Bias Vs Missingness And Delta (Scenario A)

	scenario	p_miss	delta	empirical_bias	theoretical_bias	abs_diff	N	sigma
1	A_constant_delta	0.05	0.0	-8.8e-05	-0.0	8.8e-05	200000	0.5
2	A_constant_delta	0.1	0.0	-3.1e-05	-0.0	3.1e-05	200000	0.5
3	A_constant_delta	0.2	0.0	-0.000655	-0.0	0.000655	200000	0.5
4	A_constant_delta	0.3	0.0	-0.000138	-0.0	0.000138	200000	0.5
5	A_constant_delta	0.5	0.0	8e-05	-0.0	8e-05	200000	0.5
6	A_constant_delta	0.05	0.25	-0.012719	-0.0125	0.000219	200000	0.5
7	A_constant_delta	0.1	0.25	-0.025055	-0.025	5.5e-05	200000	0.5
8	A_constant_delta	0.2	0.25	-0.050028	-0.05	2.8e-05	200000	0.5
9	A_constant_delta	0.3	0.25	-0.074206	-0.075	0.000794	200000	0.5
10	A_constant_delta	0.5	0.25	-0.123969	-0.125	0.001031	200000	0.5
11	A_constant_delta	0.05	0.5	-0.025321	-0.025	0.000321	200000	0.5
12	A_constant_delta	0.1	0.5	-0.050149	-0.05	0.000149	200000	0.5
13	A_constant_delta	0.2	0.5	-0.099986	-0.1	1.4e-05	200000	0.5
14	A_constant_delta	0.3	0.5	-0.150284	-0.15	0.000284	200000	0.5
15	A_constant_delta	0.5	0.5	-0.2505	-0.25	0.0005	200000	0.5
16	A_constant_delta	0.05	1.0	-0.050261	-0.05	0.000261	200000	0.5
17	A_constant_delta	0.1	1.0	-0.099706	-0.1	0.000294	200000	0.5
18	A_constant_delta	0.2	1.0	-0.198904	-0.2	0.001096	200000	0.5
19	A_constant_delta	0.3	1.0	-0.2997	-0.3	0.0003	200000	0.5
20	A_constant_delta	0.5	1.0	-0.500358	-0.5	0.000358	200000	0.5

Bias From Correlation Of Pi(X) With Delta(X) (Scenario B)

	scenario	a_slope	pbar_target	p_miss_empirical	d_amplitude	empirical_bias	theoretical_bias	abs_diff	corr_pi_Delta	N	sigma
1	B_corr_pi_and_Delta	-0.4	0.3	0.300117	1.0	0.034726	0.033487	0.00124	-1.0	300000	0.5
2	B_corr_pi_and_Delta	-0.2	0.3	0.299853	1.0	0.017023	0.016833	0.00019	-1.0	300000	0.5
3	B_corr_pi_and_Delta	0.0	0.3	0.299223	1.0	0.00059	0.00018	0.00041	0.0	300000	0.5
4	B_corr_pi_and_Delta	0.2	0.3	0.299633	1.0	-0.015833	-0.016473	0.00064	1.0	300000	0.5
5	B_corr_pi_and_Delta	0.4	0.3	0.300333	1.0	-0.033399	-0.033126	0.000273	1.0	300000	0.5


Scenario A: Bias ≈ - p × δ (MCAR-like, constant Δ)
Bias of MAR mean estimator
0.0
−0.1
−0.2
−0.3
−0.4
−0.5
0.1
0.2
0.3
0.4
0.5
Missingness fraction p = P(R=0)
empirical (δ=0.0)
theory (δ=0.0)
empirical (δ=0.25)
theory (δ=0.25)
empirical (δ=0.5)
theory (δ=0.5)
empirical (δ=1.0)
theory (δ=1.0)


Scenario B: Mean-zero Δ but bias from cov{π(X), Δ(X)} ≠ 0
Bias of MAR mean estimator
0.03
0.02
0.01
0.00
−0.01
−0.02
−0.03
−0.4
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
0.4
Slope a controlling π(X)–Δ(X) correlation
empirical
theory

Here you go—two small sims that make the bias identity

$$textBias(\hat{\mu}_{\text{MAR}}) \approx -E[\pi(X)\,\Delta(X)]$$

totally concrete.

- I displayed two tables to your workspace:
    - “Bias vs missingness and delta (Scenario A)” — MCAR-like $\pi(X) = p$ and constant $\Delta(X) = \delta$.
    - “Bias from correlation of pi(X) with Delta(X) (Scenario B)” — Δ(X) mean-zero but correlated with π(X).
- Two quick reads from the plots:
    - **Scenario A (constant Δ):** empirical points (circles) sit on the dashed theory lines: bias scales linearly in p and in δ (≈ −p δ).
    - **Scenario B (mean-zero Δ\Delta):** when π(X)\pi(X) tilts with X (slope a ≠ 0), you get bias even though $E[\Delta(X)] = 0$; the sign flips with the direction of the tilt, and matches $-E[\pi(X)\Delta(X)]$.

If you want, I can adapt this to your estimand (e.g., a treatment effect) or swap in a non-linear Δ(X) to show cases where the bias cancels.

```
# Simple simulation to demonstrate the bias formula:
#   Bias(MAR) = - E[ pi(X) * Delta(X) ]
#
# We'll show two scenarios:
#   A) MCAR-like missingness with constant pi(X)=p and constant Delta(X)=delta.
#      Prediction: bias ≈ - p * delta (linear in missingness and delta).
#   B) pi(X) varies with X and Delta(X) has mean zero but correlates with pi(X).
#      Prediction: bias tracks -E[pi(X) * Delta(X)] and can be nonzero even when E[Delta(X)]=0.


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from caas_jupyter_tools import display_dataframe_to_user


rng = np.random.default_rng(42)


def m1(x):
    # Observed-pattern conditional mean E[Y | X=x, R=1]
    return 1.0 + 2.0 * x


def simulate_bias_constant_delta(N=200_000, p_list=(0.05,0.1,0.2,0.3,0.5), delta_list=(0.0,0.25,0.5,1.0), sigma=0.5):
    rows = []
    for delta in delta_list:
        # Pre-generate X once per delta to keep variance low across p
        x = rng.uniform(0.0, 1.0, size=N)
        mu1 = m1(x)
        for p in p_list:
            # pi(X) = p (prob of being missing, R=0)
            pi = np.full(N, p)
            # Draw R: 0 = missing, 1 = observed
            r = rng.binomial(1, 1.0 - pi)
            # Construct Delta(X): constant shift
            Delta = np.full(N, delta)
            # Generate Y given (X,R)
            eps = rng.normal(0.0, sigma, size=N)
            y = np.where(r==1, mu1 + eps, mu1 + Delta + eps)
            # True mean
            mu_true = y.mean()
            # MAR estimator (impute missing with m1(X))
            y_imp = np.where(r==1, y, mu1)  # equivalent to setting Delta=0 for missing
            mu_mar = y_imp.mean()
            empirical_bias = mu_mar - mu_true
            # Theoretical bias: - E[ pi(X) * Delta(X) ]
            theor_bias = - np.mean(pi * Delta)
            rows.append({
                "scenario": "A_constant_delta",
                "p_miss": p,
                "delta": delta,
                "empirical_bias": empirical_bias,
                "theoretical_bias": theor_bias,
                "abs_diff": abs(empirical_bias - theor_bias),
                "N": N,
                "sigma": sigma
            })
    return pd.DataFrame(rows)


def simulate_bias_correlation(N=300_000, pbar=0.30, a_list=(-0.4,-0.2,0.0,0.2,0.4), d=1.0, sigma=0.5):
    """
    Let pi(X) = clip(pbar + a*(X - 0.5), 0.01, 0.99).
    Let Delta(X) = d*(X - 0.5) so E[Delta(X)]=0 but it correlates with pi(X) when a != 0.
    """
    rows = []
    x = rng.uniform(0.0, 1.0, size=N)
    mu1 = m1(x)
    Delta = d * (x - 0.5)
    for a in a_list:
        pi = np.clip(pbar + a * (x - 0.5), 0.01, 0.99)  # missing prob
        r = rng.binomial(1, 1.0 - pi)  # 1=observed
        eps = rng.normal(0.0, sigma, size=N)
        y = np.where(r==1, mu1 + eps, mu1 + Delta + eps)
        mu_true = y.mean()
        y_imp = np.where(r==1, y, mu1)  # MAR imputation
        mu_mar = y_imp.mean()
        empirical_bias = mu_mar - mu_true
        theor_bias = - np.mean(pi * Delta)
        rows.append({
            "scenario": "B_corr_pi_and_Delta",
            "a_slope": a,
            "pbar_target": pbar,
            "p_miss_empirical": np.mean(1 - r),
            "d_amplitude": d,
            "empirical_bias": empirical_bias,
            "theoretical_bias": theor_bias,
            "abs_diff": abs(empirical_bias - theor_bias),
            "corr_pi_Delta": np.corrcoef(pi, Delta)[0,1],
            "N": N,
            "sigma": sigma
        })
    return pd.DataFrame(rows)

# Run simulations
df_A = simulate_bias_constant_delta()
df_B = simulate_bias_correlation()


# Display tables
display_dataframe_to_user("Bias vs missingness and delta (Scenario A)", df_A.round(6))
display_dataframe_to_user("Bias from correlation of pi(X) with Delta(X) (Scenario B)", df_B.round(6))


# Plot 1: Scenario A — bias vs missingness for multiple deltas with theoretical overlay
plt.figure(figsize=(7,5))
for delta in sorted(df_A['delta'].unique()):
    sub = df_A[df_A['delta']==delta].sort_values('p_miss')
    # empirical
    plt.plot(sub['p_miss'], sub['empirical_bias'], marker='o', label=f"empirical (δ={delta})")
    # theoretical
    plt.plot(sub['p_miss'], sub['theoretical_bias'], linestyle='--', label=f"theory (δ={delta})")
plt.axhline(0, linewidth=1)
plt.title("Scenario A: Bias ≈ - p × δ (MCAR-like, constant Δ)")
plt.xlabel("Missingness fraction p = P(R=0)")
plt.ylabel("Bias of MAR mean estimator")
plt.legend()
plt.tight_layout()
plt.show()


# Plot 2: Scenario B — bias vs slope 'a' (correlation strength between pi and Delta)
plt.figure(figsize=(7,5))
subB = df_B.sort_values('a_slope')
```

```
plt.plot(subB['a_slope'], subB['empirical_bias'], marker='o', label="empirical")

plt.plot(subB['a_slope'], subB['theoretical_bias'], linestyle='--', label="theory")

plt.axhline(0, linewidth=1)

plt.title("Scenario B: Mean-zero Δ but bias from cov{π(X), Δ(X)} ≠ 0")

plt.xlabel("Slope a controlling π(X)–Δ(X) correlation")

plt.ylabel("Bias of MAR mean estimator")

plt.legend()

plt.tight_layout()

plt.show()


# Show a tiny sample of rows inline (text) so the user sees something even if they skip the full tables.

df_A.head(10).round(6), df_B.round(6)
```

Comments: The AI gives us a lot of information to take in, but for the most part the simulations seem intuitive and reflective of the bias formula. That said, there is not a strong connection being made to what $\Delta(X)$ is for this simulation – it is just treated as a constant throughout.

The introduction of correlation considerations between $\Delta(X)$ and $\pi(X)$ feel a bit unnecessary here because they are not explained in detail here or in the previous responses, so we are left with a more murky picture of what's going on in table/scenario B. Still, it is impressive how much the AI was able to produce to visualize the bias formula in so little time. The plots look correct, and scenario A's plot shows clearly how scaling up $\Delta(X)$ and the missingness fraction $\pi(X)$ causes increasing negative bias.

It is not easy to tell how the bias is estimated empirically without sifting through code, so I explore this in the next prompt.

Q: How are the empirical estimates of the bias obtained for this simulation?

Short version: I compute the population mean both **with** the MNAR truth and **with** a MAR-style imputation, then take their difference.

Here's the exact recipe the code uses.

Objects

- $X \sim \text{Unif}(0,1)$
- $m_1(x) = E(Y \mid X = x, R = 1) = 1 + 2x$
- $\Delta(x) = m_0(x) - m_1(x)$ (the pattern–mixture departure)
- $\pi(x) = P(R = 0 \mid X = x)$ (missingness prob)
- $\varepsilon \sim N(0, \sigma^2)$

Generate the "true" data (including the unobserved Y's)

For each ii:

- Draw X_i, compute $m_{1i} = m_1(X_i)$.
- Draw $R_i \sim \text{Bernoulli}(1 - \pi(X_i))$ (so R=1 means observed).
- Draw ε_i.

- Generate the latent outcome under MNAR:
$$Y_i = \begin{cases} m_{1i} + \varepsilon_i, & R_i = 1 \\ m_{1i} + \Delta(X_i) + \varepsilon_i, & R_i = 0 \end{cases}$$

Even when $R_i = 0$ (missing), we still generate Y_i because it's a simulation.

Empirical "true" mean:

$$\mu_{\text{true}} = \frac{1}{N}\sum_{i=1}^{N} Y_i$$
.

Construct the MAR estimator (what MI/FIML targets under ignorability)

Impute missing values **as if** $\Delta \equiv 0$:

$$Y_i^{\text{imp}} = \begin{cases} Y_i, & R_i = 1 \\ m_{1i}, & R_i = 0 \end{cases}$$

Then

$$\mu_{\text{MAR}} = \frac{1}{N}\sum_{i=1}^{N} Y_i^{\text{imp}}$$
.

Empirical bias (what's in the table)

$\text{empirical\_bias} = \mu_{\text{MAR}} - \mu_{\text{true}}$.

With a little algebra,

$$\mu_{\text{MAR}} - \mu_{\text{true}} = -\frac{1}{N}\sum_{i=1}^{N}(1 - R_i)\left[\Delta(X_i) + \varepsilon_i\right] \xrightarrow{E} -E\big[\pi(X)\,\Delta(X)\big]$$
,

because $E[(1-R)\varepsilon \mid X] = \pi(X)\,E(\varepsilon \mid X) = 0$. That's why the points line up with the dashed "theory" curves in the plots.

“Theoretical” line in the plots

Since in simulation we **know** π(X) and Δ(X), I also compute

$$\text{theoretical\_bias} = -\frac{1}{N}\sum_{i=1}^{N} \pi(X_i)\,\Delta(X_i)$$

,

which is the sample version of $-E[\pi(X)\Delta(X)]$. The column `abs_diff` in the tables is the tiny Monte Carlo gap between the empirical bias and this theoretical value (kept small by using large N).

Comments: The AI gives a much clearer picture of what’s going on in our simulation and how the various bias quantities are being computed. The answer is clear in almost all parts, though it does get a bit “lazy” when computing the empirical bias and shows both conditional and unconditional expectations, leveraging the tower property (law of total expectation) without saying so; I clarify this in the next response.

`Q: How come you use conditional expectation to justify E[(1-R)ε|X]=π(X)E(ε|X)=0 but the term -E[π(X)Δ(X)] is not a conditional expectation?`

Great catch. The two steps both use the tower property; I was just informal with notation in the first. Here's the precise, side-by-side derivation.

Setup (pattern–mixture parametrization)

Let

$$m_r(x) := E(Y \mid X = x, R = r), \qquad \Delta(x) := m_0(x) - m_1(x), \qquad \pi(x) := P(R = 0 \mid X = x).$$

Write Y as

$$Y = m_1(X) + \underbrace{(1 - R)\Delta(X)}_{\text{pattern shift}} + \varepsilon,$$

where $\varepsilon := Y - E(Y \mid X, R)$ so that, **by construction,**

$$E(\varepsilon \mid X, R) = 0.$$

(Equivalently, mrm_r already absorbs any conditional mean; ε is the mean-zero residual within each pattern.)

The MAR-imputation target fills in missing values with $m_1(X)$, i.e.

$$Y^{\text{imp}} = \begin{cases} Y, & R = 1, \\ m_1(X), & R = 0, \end{cases} \qquad \Rightarrow \qquad Y^{\text{imp}} - Y = -(1 - R)\{\Delta(X) + \varepsilon\}.$$

Therefore the **bias in the marginal mean** is

$$\text{Bias} = E\big[Y^{\text{imp}} - Y\big] = -\,E\big[(1 - R)\Delta(X)\big] \; - \; E\big[(1 - R)\varepsilon\big].$$

Why the ε\varepsilon term vanishes (the careful way)

Use the tower property **conditioning on (X,R)**:

$$E\big[(1-R)\varepsilon\big]=E\big[\,E\big((1-R)\varepsilon\mid X,R\big)\big]=E\big[(1-R)\,E(\varepsilon\mid X,R)\big]=0,$$

because $E(\varepsilon\mid X,R)=0$ by definition.
Note: this does **not** assume any independence between ε and R; it's built into how we defined m_r and ε.

(What I wrote earlier, $E(\#\#\#)E[(1-R)\varepsilon\mid X]=\pi(X)E(\varepsilon\mid X)$, implicitly uses $E(\varepsilon\mid X,R)=E(\varepsilon\mid X)=0$. The tighter argument above avoids that extra step and is preferable.)

Why the remaining term becomes $-E[\pi(X)\Delta(X)]$

Now apply the tower property **conditioning on X** to the remaining piece:

$$E\big[(1-R)\Delta(X)\big]=E\big[\,E\big((1-R)\Delta(X)\mid X\big)\big]=E\big[\Delta(X)\,E(1-R\mid X)\big]=E\big[\pi(X)\,\Delta(X)\big].$$

No independence is assumed here either; we only used that Δ(X) is a function of X and $E(1-R\mid X)=\pi(X)$.

Putting it together,

$$\boxed{\text{Bias}=-\,E\big[\pi(X)\,\Delta(X)\big].}$$

Same result via $E(Y\mid X)$

You can also start from

$$E(Y\mid X)=(1-\pi(X))\,m_1(X)+\pi(X)\,m_0(X)=m_1(X)+\pi(X)\Delta(X),$$

so $E(Y)=E[m_1(X)]+E[\pi(X)\Delta(X)]$.
The MAR plug-in targets $E[m_1(X)]$, hence the same bias $-E[\pi(X)\Delta(X)]$.

Summary:

- The ε-term is killed by conditioning on (X,R) and using $E(\varepsilon\mid X,R)=0$.

- The πΔ-term *does* come from a conditional expectation step—conditioning on X—and then "unconditioning" to write it as an unconditional expectation over X.

Comments: It's a bit tricky to see the Y decomposition at first (that part could use more detail), but otherwise I found this answer to be a helpful formalization of the claims made in the previous response that made the theory clearer. This response used notation effectively and overall I see no technical issues with it.

Tab 5

Q: Discuss the applicability of various MICE methods. What are their strengths and weaknesses?

Here's a structured discussion of the **applicability of different MICE (Multivariate Imputation by Chained Equations) methods**, highlighting their **strengths and weaknesses**:

---

1. Predictive Mean Matching (pmm)

- **How it works:** Fits a regression model, predicts missing values, and then matches each missing case to an observed donor with a similar predicted value. The observed donor's actual value is used as the imputation.
- **Strengths:**
    - Preserves the distribution of the data (avoids implausible imputations).
    - Handles skewed and non-normal data better than parametric methods.
    - Robust to model misspecification because imputations are drawn from observed values.
- **Weaknesses:**
    - Performance depends on the quality of the predictive model.
    - Limited extrapolation ability: missing values outside the observed range cannot be imputed reasonably.
    - Sensitive to the donor pool size (too few donors → instability; too many → loss of local accuracy).

---

2. Norm (Bayesian Linear Regression with Normal Posterior Approximation)

- **How it works:** Uses linear regression and Bayesian draws for regression coefficients and residual variance. Missing values are imputed as draws from this predictive distribution.
- **Strengths:**
    - Theoretically principled under MAR and when the linear model is correct.
    - Incorporates parameter uncertainty via Bayesian draws.
    - Provides unbiased estimates if the model assumptions hold (normality, linearity, homoscedasticity).
- **Weaknesses:**
    - Sensitive to model misspecification.
    - Can produce implausible imputations (outside the observed range).
    - Assumes approximately normal residuals, which may not hold in real data.

---

3. Norm.nob (Classical Linear Regression without Bayesian Draws)

- **How it works:** Similar to `norm`, but imputations use fixed regression estimates without parameter uncertainty.

- **Strengths:**
    - Simpler and faster than `norm`.
    - Still reasonable if sample sizes are large and regression model is well-specified.
- **Weaknesses:**
    - Fails to account for imputation uncertainty → underestimates variability in downstream analyses.
    - Can lead to overly optimistic standard errors and coverage issues.

---

4. Cart (Classification and Regression Trees)

- **How it works:** Uses recursive partitioning to predict missing values from observed predictors.
- **Strengths:**
    - Captures nonlinear relationships and interactions automatically.
    - Flexible for complex data structures where linearity doesn't hold.
    - Works well with mixed data types (continuous, categorical).
- **Weaknesses:**
    - Tends to overfit small donor pools unless tuned.
    - Computationally intensive for large datasets.

    - Imputations can be less stable if tree depth or pruning is not controlled.

---

5. Logreg / Polyreg (Logistic & Polytomous Regression for Categorical Data)

- **How they work:** Fit regression models appropriate for binary (`logreg`) or multi-category (`polyreg`) variables.
- **Strengths:**
    - Ensures imputations respect categorical boundaries.
    - Well-suited when relationships are indeed logistic/multinomial.
    - Good theoretical justification under MAR if the model is correct.
- **Weaknesses:**
    - Poor performance with small sample sizes or sparse categories.
    - Sensitive to model misspecification (e.g., non-linear effects not modeled).
    - Can produce separation issues in logistic regression with rare outcomes.

---

6. Rf (Random Forests)

- **How it works:** Uses random forests as predictive models for imputations.

- **Strengths:**
    - Flexible, nonparametric, captures nonlinearities and interactions.
    - Robust to model misspecification compared to linear methods.
    - Handles mixed data types naturally.
- **Weaknesses:**
    - Computationally expensive with large data.
    - Can produce biased imputations if variables with missingness are weakly related to others.
    - Does not inherently account for uncertainty well → may underestimate variance.

---

7. Mean / Mode Imputation (not usually recommended)

- **How it works:** Replaces missing values with the mean (continuous) or mode (categorical).
- **Strengths:**
    - Very simple to implement.
    - Preserves sample size.
- **Weaknesses:**

    - Severely biases associations (attenuates correlations, regression slopes).
    - Underestimates variability.
    - Only acceptable in extremely simple exploratory contexts.

---

8. 2L Methods (e.g., 2l.norm, 2l.pmm)

- **How they work:** Specialized for multilevel/hierarchical data, accounting for clustering.
- **Strengths:**
    - Maintains within- and between-cluster variability.
    - Avoids artificially inflating precision in multilevel models.
- **Weaknesses:**
    - More complex and computationally heavy.
    - Sensitive to misspecification of the random-effects structure.

Summary Table

Method	Best For	Strengths	Weaknesses
pmm	Skewed/realistic distributions	Robust, preserves distribution	Poor extrapolation

norm	Linear normal models	Theoretically principled	Implausible values if assumptions fail
norm.nob	Large n, quick imputations	Fast, simple	Underestimates variance
cart	Nonlinear/mixed data	Captures interactions	Overfitting, unstable
logreg/polyreg	Categorical vars	Proper handling of categories	Sparse data issues
rf	Complex nonlinear data	Flexible, robust	Computationally heavy
mean/mode	Almost never	Simple	Severe bias
2L methods	Multilevel data	Proper variance partitioning	Complex, slower

---

👉 Would you like me to also **create an R simulation comparing a few of these (say, norm, pmm, cart, rf) under different data-generating scenarios** so we can see the practical differences in bias, variance, and coverage?

Comments: The AI gives us a very long and thorough discussion of various methods that exist within MICE, using R terminology since it is perhaps implied that we are referring to the R package. Unfortunately, the AI is mistaken on norm.nob, and includes mean/mode imputation which I would not consider MICE methods. The mention of so-called "2L" methods is new, highlighting how expansive this response was. Most of the claims made by the AI here are correct and noteworthy, but a few of them are a bit more subtle and hard to verify (for example, Rf "does not inherently account for uncertainty well").

Q: Your explanation of norm.nob seems wrong.

You’re right to flag this—my wording was imprecise. Here’s the precise picture:

- **What `norm.nob` actually does (in `mice`)**
  It’s **stochastic regression imputation**: fit OLS once (no Bayesian draw of parameters), then impute $y_{\text{mis}} = x_{\text{mis}}\hat{\beta} + \varepsilon$∧* with ε∧* $\sim \mathcal{N}(0, \hat{\sigma}^2)$.
  So it **does add random residual noise**, but it **does not** propagate **parameter uncertainty** via posterior draws of (β, σ^2). In MICE terms this is “predict + noise,” while `norm` is “predict + noise + parameter uncertainty.” (Stef van Buuren)

- **Why this matters**
  Because `norm.nob` fixes $\hat{\beta}, \hat{\sigma}^2$, the **between-imputation variance B** is too small. Rubin’s total variance $T = W + (1 + 1/m)B$ is then underestimated, yielding **CIs that are too narrow** (improper MI), unless nn is very large so that ignoring parameter uncertainty is negligible. This is exactly how the method is described in the docs and texts. (RDocumentation, R Project Search, jeffgill.org)

- **Contrast with nearby methods**

  - `norm`: Bayesian linear regression; draws (β,σ) from the posterior each time → **proper MI** under model correctness. (Welcome to amices)

  - `norm.predict`: deterministic regression prediction (no residual draw) → **strongly discouraged**; seriously biases associations and underestimates variance. (Welcome to amices)

    - `norm.boot`: uses **bootstrap** to approximate parameter uncertainty (a pragmatic alternative to full Bayes). (Welcome to amices)

- **Takeaway**
  `norm.nob` is **not** the deterministic one; it is **stochastic** but **ignores parameter uncertainty**, which typically **understates SEs/CI widths**. It can be acceptable for **very large nn** or quick exploratory runs, but for inferential work you generally prefer `norm` (or `norm.boot`) or a semi-parametric choice like `pmm`. (jeffgill.org)

If you'd like, I can show a tiny simulation that compares CI coverage for `norm`, `norm.nob`, and `norm.predict` so the difference is visible.

Comments: The AI correctly identifies that norm.nob is not deterministic (like norm.predict) and correctly clarifies how it propagates uncertainty into the imputation. It claims that the between-imputation variance B is too small because of this, but this claim is one that could easily be right and could easily be wrong; it needs to be proven or at least demonstrated in simulation first.

Q: Can you prove that the between-imputation variance B is too small under norm.nob?

Great question. Here's a clean way to see why **the between–imputation variance BB is too small** under `norm.nob`.

1) General argument (law of total variance)

Let $Y = (Y_{\text{obs}}, Y_{\text{mis}})$. Let $\hat{Q} = \hat{Q}(Y)$ be your completed–data estimator (e.g., a mean, a regression coefficient, etc.). In multiple imputation, the between–imputation variance estimates

$$B \;\approx\; \text{Var}_{\text{imp}}\big(\hat{Q}^{(k)} \,\big|\, Y_{\text{obs}}\big),$$

the variance across completed datasets generated from the **imputation distribution** for Y_{mis} given Y_{obs}.

- **Proper MI** draws Y_{mis} from the **posterior predictive distribution**

$$p(Y_{\text{mis}} \mid Y_{\text{obs}}) \;=\; \int p(Y_{\text{mis}} \mid \theta, Y_{\text{obs}})\; p(\theta \mid Y_{\text{obs}})\; d\theta.$$

- **`norm.nob`** instead conditions on a **plug-in estimate** $\hat{\theta}$ (OLS point estimate), drawing

$$Y_{\text{mis}} \sim p\Big(Y_{\text{mis}} \mid \hat{\theta}, Y_{\text{obs}}\Big),$$

i.e., no draw of θ from its posterior.

Consider the variance of Q^\hat Q across imputations under the **proper** mechanism. Using the law of total variance with respect to $\theta \mid Y_{\text{obs}}$,

$$\mathrm{Var}_{\text{proper}}(\widehat{Q}|Y_{\text{obs}})=\underbrace{\mathbb{E}_{\theta|Y_{\text{obs}}}\left[\mathrm{Var}(\widehat{Q}|\theta,Y_{\text{obs}})\right]}_{\text{within}-(\theta)\text{ term}}+\underbrace{\mathrm{Var}_{\theta|Y_{\text{obs}}}\left(\mathbb{E}[\widehat{Q}|\theta,Y_{\text{obs}}]\right)}_{\text{between}-(\theta)\text{ term}}.\ (1)$$

Under **`norm.nob`**, the imputer **fixes** $\theta = \hat{\theta}$, hence

$$\mathrm{Var}_{\text{nob}}\big(\hat{Q} \mid Y_{\text{obs}}\big) = \mathrm{Var}\big(\hat{Q} \mid \hat{\theta}, Y_{\text{obs}}\big). \qquad (2)$$

Comparing (1) and (2):

$$\mathrm{Var}_{\text{proper}}\big(\hat{Q} \mid Y_{\text{obs}}\big) = \mathbb{E}_{\theta|Y_{\text{obs}}}\Big[\mathrm{Var}\big(\hat{Q} \mid \theta, Y_{\text{obs}}\big)\Big] \ + \ \mathrm{Var}_{\theta|Y_{\text{obs}}}\Big(\mathbb{E}[\hat{Q} \mid \theta, Y_{\text{obs}}]\Big) \ \geq \ \mathrm{Var}\big(\hat{Q} \mid \hat{\theta}, Y_{\text{obs}}\big),$$

with equality only if the second (between-θ) term is zero (i.e., $\mathbb{E}[\hat{Q} \mid \theta, Y_{\text{obs}}]$ **does not** depend on θ) **and** $\mathrm{Var}(\hat{Q} \mid \theta, Y_{\text{obs}})$ **is flat enough that taking** $\theta = \hat{\theta}$ equals the θ-average—situations that occur only in degeneracies or as $n \to \infty$.

Therefore, as $m \to \infty$,

$$\underbrace{B_{\text{nob}}}_{\text{what `norm.nob` estimates}} \quad = \ \mathrm{Var}_{\text{nob}}\big(\hat{Q} \mid Y_{\text{obs}}\big) \ < \ \mathrm{Var}_{\text{proper}}\big(\hat{Q} \mid Y_{\text{obs}}\big) \ = \quad \underbrace{B_{\text{proper}}}_{\text{target for proper MI}},$$

i.e., **`norm.nob`** systematically omits the between-θ component, so B is too small (except in the large-nn limit or pathological cases).

---

2) Concrete linear-model example (visible extra term)

Suppose we impute Y from a linear model

$$Y = X^\top \beta + \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, \sigma^2),$$

with some entries of Y missing and all X observed. Let M index the r missing cases and X_M be their design rows stacked. Consider estimating the **overall mean** $\mu = \frac{1}{n}\sum_{i=1}^{n} Y_i$ by $\hat{\mu}^{(k)}$ from each completed dataset.

- Under **`norm`** **(proper)**: draw $\beta^{(k)} \sim p(\beta \mid Y_{\text{obs}})$ (approximately $\mathcal{N}(\hat{\beta}, V_\beta)$) and residuals $\varepsilon_M^{(k)}$.

  $$Y_M^{(k)} = X_M \beta^{(k)} + \varepsilon_M^{(k)}, \qquad \hat{\mu}^{(k)} = \frac{1}{n}\left(\sum_{i \notin M} Y_i + \mathbf{1}_r^\top Y_M^{(k)}\right).$$

  The across-imputation variance is

  $$\mathrm{Var}_{\text{proper}}\big(\hat{\mu}^{(k)} \mid Y_{\text{obs}}\big) = \underbrace{\frac{1}{n^2}\,\mathbf{1}_r^\top \big(X_M V_\beta X_M^\top\big)\mathbf{1}_r}_{\text{parameter-uncertainty term}} + \underbrace{\frac{r}{n^2}\,\sigma^2}_{\text{residual term}}.$$

- Under **`norm.nob`**: fix $\beta = \hat{\beta}$, draw only residuals:

  $$Y_M^{(k)} = X_M \hat{\beta} + \varepsilon_M^{(k)}, \qquad \Rightarrow \qquad \mathrm{Var}_{\text{nob}}\big(\hat{\mu}^{(k)} \mid Y_{\text{obs}}\big) = \frac{r}{n^2}\,\hat{\sigma}^2.$$

Hence

$$\mathrm{Var}_{\text{proper}}\big(\hat{\mu}^{(k)} \mid Y_{\text{obs}}\big) - \mathrm{Var}_{\text{nob}}\big(\hat{\mu}^{(k)} \mid Y_{\text{obs}}\big) = \frac{1}{n^2}\,\mathbf{1}_r^\top \big(X_M V_\beta X_M^\top\big)\mathbf{1}_r \geq 0,$$

with **strict positivity** unless $V_\beta = \mathbf{0}$ (i.e., infinite information) or $X_M^\top \mathbf{1}_r = \mathbf{0}$ in a contrived way. Thus the extra, **non-negative "parameter-uncertainty" term is missing under `norm.nob`**, making B too small.

The same calculation (via a first-order linearization) extends to smooth functionals Q and to regression coefficients as targets: the across-imputation variance under proper draws acquires an additive V_θ-driven term that **`norm.nob` discards**.

---

3) Consequences

- Rubin's total variance $T = \bar{W} + (1 + 1/m)B$ will be **underestimated** because B is too small.
- Confidence intervals are **too narrow** and coverage falls **below nominal**.
- The gap shrinks with increasing n (posterior $V_\theta \to 0$), but at realistic sample sizes it can be material.

Bottom line: `norm.nob` is *stochastic* (it adds residual noise) but **improper** because it **fails to propagate parameter uncertainty**. Formally, it replaces the posterior predictive variance by the conditional variance at $\hat{\theta}$, thereby dropping the between-θ component and making B too small.

Comments: See next response for discussion.

Q: Can you show that the parameter-uncertainty term is 0 in your normal example for norm.nob?

Note: This response renders poorly: See https://chatgpt.com/s/t_68a2023bb2a48191850df695201985c5

Yes—under `norm.nob` the **parameter-uncertainty term is exactly 0** because the imputer **does not randomize** $\theta = (\beta, \sigma^2)$ across imputations. Here's the clean derivation in the normal linear example.

Setup

Imputation model:

$$Y_i = x_i^\top \beta + \varepsilon_i, \qquad \varepsilon_i \sim \mathcal{N}(0, \sigma^2),$$

with some Y_i missing and X fully observed. Let M index the r missing cases and stack their rows into X_M. Consider the completed–data estimator of the mean

$$\hat{\mu}^{(k)} = \frac{1}{n}\left(\sum_{i \notin M} Y_i + \mathbf{1}_r^\top Y_M^{(k)}\right).$$

- **Proper MI (`norm`)**: draw $\beta^{(k)} \sim p(\beta \mid Y_{\text{obs}})$, $\sigma^{2(k)} \sim p(\sigma^2 \mid Y_{\text{obs}})$, and residuals $\varepsilon_M^{(k)}$, then
  $Y_M^{(k)} = X_M \beta^{(k)} + \varepsilon_M^{(k)}$.
- **`norm.nob`**: **fix** $\hat{\beta}$, $\hat{\sigma}^2$ from OLS (no posterior draw), then draw only residuals
  $Y_M^{(k)} = X_M \hat{\beta} + \varepsilon_M^{(k)}, \qquad \varepsilon_M^{(k)} \sim \mathcal{N}(0, \hat{\sigma}^2 I_r)$,

independently over k.

Throughout, B is the **across-imputation variance conditional on** Y_{obs}.

Law-of-total-variance decomposition

For any imputation mechanism,

$$\mathrm{Var}_{\text{imp}}(\widehat{\mu}^{(k)}|Y_{\text{obs}})=\underbrace{\mathbb{E}\Big[\mathrm{Var}\big(\mathbb{p}^{(\mathbb{k})},\mathbb{Y}_{\text{obs}}\big)\Big|\mathbb{Y}_{\text{obs}}\Big]}_{\text{residual (within-}(\theta))}+\underbrace{\mathrm{Var}\Big(\mathbb{E}\big[\mathbb{p}^{(\mathbb{k})},\mathbb{Y}_{\text{obs}}\big]\Big|\mathbb{Y}_{\text{obs}}\Big]}_{\text{parameter-uncertainty (between-}(\theta))}.\text{[\textbackslash tag?]}^*$$

Under **`norm.nob`**, the “posterior” for $\theta \mid Y_{\text{obs}}$ is **degenerate** at $\hat{\theta} = (\hat{\beta}, \hat{\sigma}^2)$. Thus the second term in (*) is the variance of a **constant**, hence zero. One can also show this directly:

Direct calculation under `norm.nob`

With θ fixed at $\hat{\theta}$,

$$\hat{\mu}^{(k)} = \frac{1}{n}\left(\sum_{i \notin M} Y_i + \mathbf{1}_r^\top X_M \hat{\beta}\right) + \frac{1}{n}\,\mathbf{1}_r^\top \varepsilon_M^{(k)}.$$

- **Across-imputation expectation (conditional on YobsY_{\text{obs}})**:
  $$\mathbb{E}_{\text{imp}}[\hat{\mu}^{(k)} \mid Y_{\text{obs}}] = \frac{1}{n}\left(\sum_{i \notin M} Y_i + \mathbf{1}_r^\top X_M \hat{\beta}\right),$$
  which is **deterministic across imputations** (no β-draw).

- **Across-imputation variance (conditional on** Y_{obs}**)**:
  $$\mathrm{Var}_{\text{imp}}\big(\hat{\mu}^{(k)} \mid Y_{\text{obs}}\big) = \mathrm{Var}\left(\frac{1}{n}\,\mathbf{1}_r^\top \varepsilon_M^{(k)}\right) = \frac{1}{n^2}\,\mathbf{1}_r^\top\,(\hat{\sigma}^2 I_r)\,\mathbf{1}_r = \frac{r}{n^2}\,\hat{\sigma}^2.$$

Comparing to (*), the **parameter-uncertainty term is zero** because $\mathbb{E}[\hat{\mu}^{(k)} \mid \theta, Y_{\text{obs}}]$ does **not vary** with θ across imputations when θ is fixed at $\hat{\theta}$.

Same conclusion for any linear functional

Let $Q = a^\top Y$ be any **linear** target (e.g., a contrast, a domain mean, or the linearization of a smooth Q). Split $a = (a_{\text{obs}}, a_M)$. Under `norm.nob`,

$$Q^{(k)} = a_{\text{obs}}^\top Y_{\text{obs}} + a_M^\top (X_M \hat{\beta}) + a_M^\top \varepsilon_M^{(k)}.$$

Hence

$$\mathbb{E}_{\text{imp}}[Q^{(k)} \mid Y_{\text{obs}}] = a_{\text{obs}}^\top Y_{\text{obs}} + a_M^\top (X_M \hat{\beta}) \quad (\text{constant in (k)}),$$

and

$$\text{Var}_{\text{imp}}(Q^{(k)} \mid Y_{\text{obs}}) = a_M^\top \hat{\sigma}^2 I_r \, a_M = \hat{\sigma}^2 \, \|a_M\|^2.$$

Again, the **parameter-uncertainty component is 0**.

Takeaway

In `norm.nob`, the imputer **does not draw** β, σ^2; equivalently, the posterior of $\theta \mid Y_{\text{obs}}$ is treated as a **point mass** at $\hat{\theta}$. Therefore, the **between-θ** piece in the MI variance decomposition vanishes, and the across-imputation variability B reflects **only residual noise**. This is exactly why `norm.nob` underestimates Rubin's $T = \bar{W} + (1 + 1/m)B$ unless V_β is negligible (very large n).

Comments: See next response for discussion.

Q: But you did not have sigma^2 hat in the Var proper computation, just sigma^2.

Good catch—you're right. In the **proper MI** case (e.g., `norm`), the residual piece should be

$$\frac{r}{n^2}\,\mathbb{E}\big[\sigma^2 \mid Y_{\text{obs}}\big],$$

not simply $\frac{r}{n^2}\sigma^2$. I was being loose with notation. Here's the corrected, side-by-side derivation.

Corrected comparison (linear-normal imputation, target = mean)

Let M index the r missing Y's and X_M be their design rows. For the completed-data mean

$$\hat{\mu}^{(k)} = \frac{1}{n}\Big(\sum_{i\notin M} Y_i + \mathbf{1}_r^\top Y_M^{(k)}\Big),$$

we have:

Proper MI (`norm`): draw $\beta^{(k)}, \sigma^{2(k)} \sim p(\beta, \sigma^2 \mid Y_{\text{obs}})$ and $\varepsilon_M^{(k)} \sim \mathcal{N}(0, \sigma^{2(k)} I)$, so

$$Y_M^{(k)} = X_M\beta^{(k)} + \varepsilon_M^{(k)}.$$

By the law of total variance,

$$\begin{aligned}\operatorname{Var}_{\text{proper}}\big(\hat{\mu}^{(k)} \mid Y_{\text{obs}}\big) &= \mathbb{E}_{\theta\mid Y_{\text{obs}}}\big[\operatorname{Var}\big(\hat{\mu}^{(k)} \mid \theta, Y_{\text{obs}}\big)\big] + \operatorname{Var}_{\theta\mid Y_{\text{obs}}}\big(\mathbb{E}[\hat{\mu}^{(k)} \mid \theta, Y_{\text{obs}}]\big) \\ &= \underbrace{\frac{r}{n^2}\,\mathbb{E}\big[\sigma^2 \mid Y_{\text{obs}}\big]}_{\text{residual term}} \;+\; \underbrace{\frac{1}{n^2}\,\mathbf{1}_r^\top\big(X_M V_\beta X_M^\top\big)\mathbf{1}_r}_{\text{parameter-uncertainty in }\beta},\end{aligned}$$

because $\mathbb{E}[\hat{\mu}^{(k)} \mid \theta, Y_{\text{obs}}] = \dfrac{1}{n}\big(\sum_{i\notin M} Y_i + \mathbf{1}_r^\top X_M\beta\big)$ depends on β but **not on** σ^2.

`norm.nob`: fix $\hat{\beta}, \hat{\sigma}^2$ (no draws), draw only $\varepsilon_M^{(k)} \sim \mathcal{N}(0, \hat{\sigma}^2 I)$:

$$\operatorname{Var}_{\text{nob}}\big(\hat{\mu}^{(k)} \mid Y_{\text{obs}}\big) = \frac{r}{n^2}\,\hat{\sigma}^2.$$

Difference:

$$\operatorname{Var}_{\text{proper}} - \operatorname{Var}_{\text{nob}} = \frac{1}{n^2}\,\mathbf{1}_r^\top\big(X_M V_\beta X_M^\top\big)\mathbf{1}_r \;+\; \frac{r}{n^2}\Big(\mathbb{E}[\sigma^2 \mid Y_{\text{obs}}] - \hat{\sigma}^2\Big).$$

- The **first term** is the missing parameter-uncertainty in β. This is the dominant gap and is **strictly non-negative**, zero only in degenerate/large-nn cases with $V_\beta = 0$ or when $X_M^\top\mathbf{1}_r = \mathbf{0}$ in a contrived setup.

- The **second term** is a small finite-sample correction from drawing σ^2 under proper MI. With common conjugate/Jeffreys priors for the Gaussian linear model, $\sigma^2 \mid Y_{\text{obs}}$ has an inverse-χ2 posterior with $\mathbb{E}[\sigma^2 \mid Y_{\text{obs}}] \;=\; \dfrac{\text{SSE}}{\nu - 2} \quad \text{where} \quad \nu = n_{\text{obs}} - p$,
  while the usual OLS estimate uses $\hat{\sigma}^2 = \text{SSE}/\nu$. Hence $\mathbb{E}[\sigma^2 \mid Y_{\text{obs}}] > \hat{\sigma}^2$ for $\nu > 2$, making the difference **even more positive**. In large n_{obs}, this gap is $O(1/\nu)$ and negligible.

So the fully corrected statement is:

$$B_{\text{nob}} \;=\; \frac{r}{n^2}\hat{\sigma}^2 \quad \text{vs} \quad B_{\text{proper}} \;=\; \frac{1}{n^2}\,\mathbf{1}_r^\top\big(X_M V_\beta X_M^\top\big)\mathbf{1}_r \;+\; \frac{r}{n^2}\,\mathbb{E}[\sigma^2 \mid Y_{\text{obs}}].$$

This makes clear **why B_{nob} is too small**: it **drops the V_β term entirely** and typically also uses a slightly smaller residual variance than the posterior mean.

Comments: As we see from these responses, the AI is comfortable getting very deep into theory and showing complex, linear algebra-based computations with ease. That said, the AI is a little sloppy in its work (perhaps because the notation is already burdensome). We see for example that a σ^2 term appears in the “proper” norm variance computation, but I’m suspicious of this because this is a random variable in Bayesian terms and because the AI then goes on to cancel it out with $\hat{\sigma}^2$ in the next line. In the third response we see an attempted correction of this which isn’t especially convincing either, because the law of total variance has been applied in a somewhat murky way and it’s unclear how we’re arriving at the expectation of $\sigma^2 \mid Y_{obs}$. Although the AI may have the right idea, these responses show where it can struggle as a teacher; even if you have the correct theory available, you need to be able to explain it clearly and cleanly, but these responses consistently leave questions open. This is partly to do with burdensome notation and partly to do with poor choice of phrasing or mathematical construction.

Q: But then I think we should have an expectation for sigma^2_hat too since it is an estimator.

(This question is intentionally misleading.)

Great point—and this comes down to **what randomness B** is meant to capture.

1) Why there's no $\mathbb{E}[\hat{\sigma}^2]$ inside B (the conditional view)

By definition in MI, the between–imputation variance is

$$B \;=\; \mathrm{Var}_{\mathrm{imp}}\big(\hat{Q}^{(k)} \mid Y_{\mathrm{obs}}\big).$$

Here we **condition on the realized** Y_{obs}. Under `norm.nob`, $\hat{\beta}$ and $\hat{\sigma}^2$ are functions of Y_{obs} only, so **once** Y_{obs} **is fixed,** $\hat{\beta}, \hat{\sigma}^2$ **are constants** across imputations. Hence, in the normal linear example for the mean,

$$B_{\mathrm{nob}} = \mathrm{Var}_{\mathrm{imp}}\left(\tfrac{1}{n}\mathbf{1}_r^\top \varepsilon_M^{(k)} \;\middle|\; Y_{\mathrm{obs}}\right) = \frac{r}{n^2}\,\hat{\sigma}^2,$$

with **no expectation** over $\hat{\sigma}^2$ because it's fixed **given** Y_{obs}.
That's exactly the sense in which the "parameter–uncertainty term" is 0 under `norm.nob`: there is **no variability from re–drawing** $\theta = (\beta, \sigma^2)$ across imputations.

2) If you want to step outside the conditional view (the unconditional/frequentist view)

You can also compare the **expected** B's over repeated samples of Y_{obs}. Then you *do* take expectations of $\hat{\sigma}^2$ etc.:

- Under `norm.nob`:

$$\mathbb{E}_{Y_{\text{obs}}}\big[B_{\text{nob}}\big] = \frac{r}{n^2}\,\mathbb{E}_{Y_{\text{obs}}}\big[\hat{\sigma}^2\big].$$

If $\hat{\sigma}^2$ is the usual unbiased OLS estimator $\text{SSE}/(n_{\text{obs}} - p)$, then $\mathbb{E}[\hat{\sigma}^2] = \sigma^2$, so

$$\mathbb{E}_{Y_{\text{obs}}}\big[B_{\text{nob}}\big] = \frac{r}{n^2}\,\sigma^2.$$

- Under **proper MI**:

$$\mathbb{E}_{Y_{\text{obs}}}\big[B_{\text{proper}}\big] = \underbrace{\frac{r}{n^2}\,\mathbb{E}_{Y_{\text{obs}}}\big[\mathbb{E}(\sigma^2 \mid Y_{\text{obs}})\big]}_{=\frac{r}{n^2}\sigma^2\ \text{by the tower property}} + \underbrace{\frac{1}{n^2}\,\mathbb{E}_{Y_{\text{obs}}}\Big[\mathbf{1}_r^\top (X_M V_\beta X_M^\top)\mathbf{1}_r\Big]}_{\ge 0}.$$

So even **in expectation over samples**, the residual pieces match (both equal $\frac{r}{n^2}\sigma^2$), and the **only difference** is the extra, non-negative **parameter-uncertainty term** involving $V_\beta = \text{Var}(\beta \mid Y_{\text{obs}})$. Therefore

$$\mathbb{E}_{Y_{\text{obs}}}\big[B_{\text{proper}}\big] - \mathbb{E}_{Y_{\text{obs}}}\big[B_{\text{nob}}\big] = \frac{1}{n^2}\,\mathbb{E}_{Y_{\text{obs}}}\Big[\mathbf{1}_r^\top (X_M V_\beta X_M^\top)\mathbf{1}_r\Big] > 0$$

Unless $V_\beta \equiv 0$ (the infinite-information/large-n limit or a degeneracy).

3) What this means

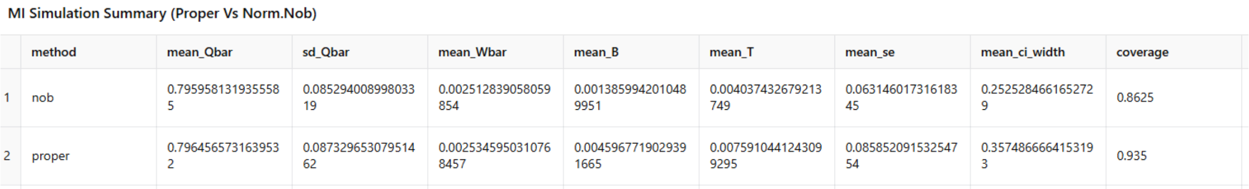
- **Conditional (what MI uses for B)**: $\hat{\sigma}^2$ is fixed given Y_{obs}, so there's no expectation of it inside B. `norm.nob` drops parameter uncertainty altogether → B too small **conditional on the data**.

- **Unconditional (frequentist averaging)**: both methods' residual terms average to $\frac{r}{n^2}\sigma^2$; the **strictly positive** V_β term remains for proper MI → $\mathbb{E}[B_{\text{proper}}] > \mathbb{E}[B_{\text{nob}}]$.

Either way of looking at it leads to the same conclusion: the shortfall of `norm.nob` is **not** about whether you take $\mathbb{E}[\hat{\sigma}^2]$; it's that `norm.nob` **never varies** β **(or σ^2) across imputations**, so it **omits** the between-θ component entirely.

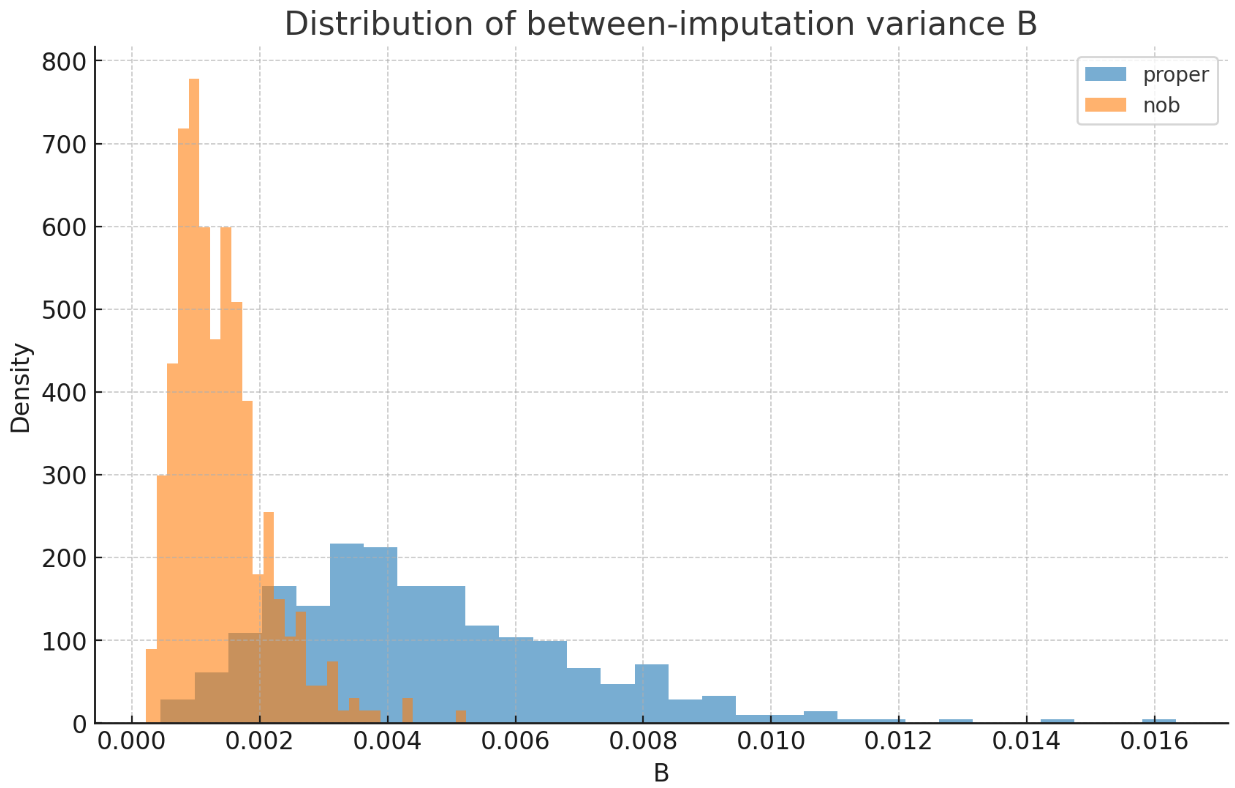
Comments: The AI succeeds in not 'taking the bait' despite the reasonable-looking concern that we take expectations for one random variable but not another. It then breaks down the details a little more thoroughly and discusses how we might end up taking expectations if we are interested in the average behavior of B. Overall this answer doesn't appear to make any egregious mistakes and does not fall into the agreeability trap of trying to force my wrong correction in the way of the actual mathematics.

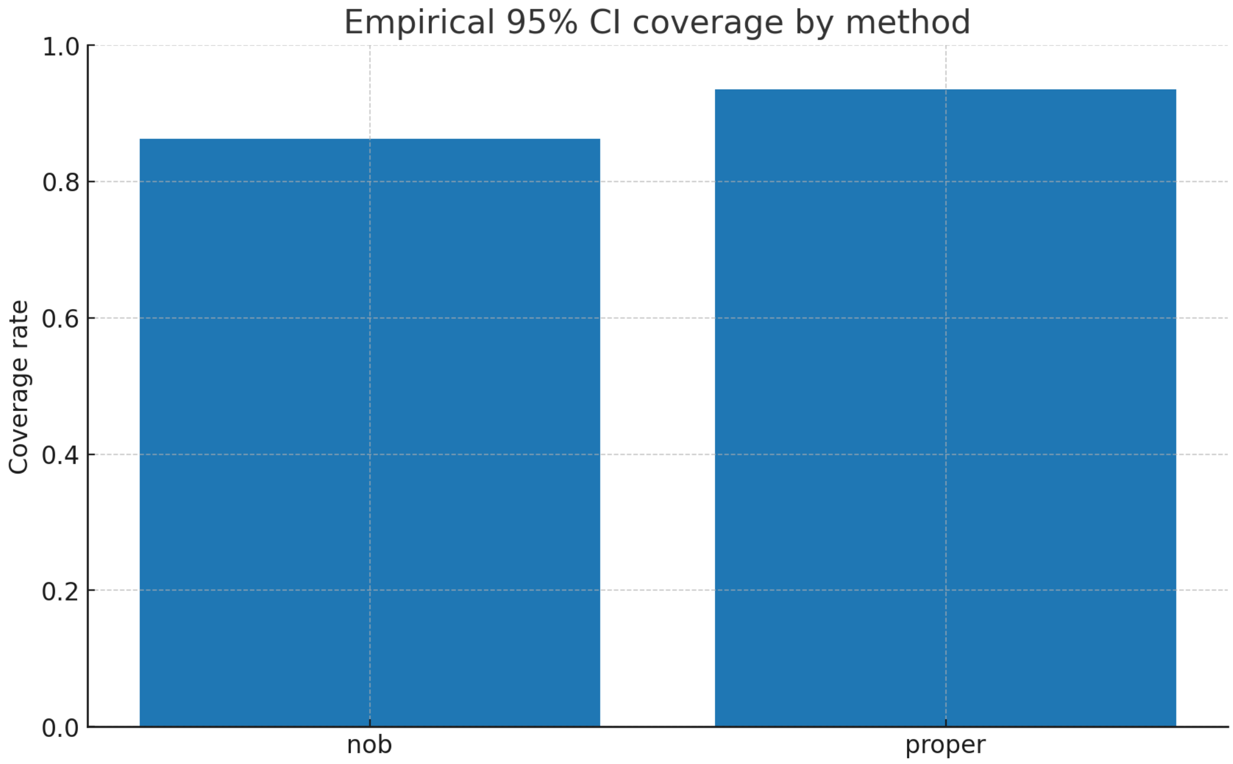
Q: Can you demonstrate the negative bias associated with norm.nob clearly via simulation?



MI Simulation Summary (Proper Vs Norm.Nob)

	method	mean_Qbar	sd_Qbar	mean_Wbar	mean_B	mean_T	mean_se	mean_ci_width	coverage
1	nob	0.7959581319355585	0.08529400899803319	0.002512839058059854	0.0013859942010489951	0.004037432679213749	0.06314601731618345	0.2525284661652729	0.8625
2	proper	0.7964565731639532	0.08732965307951462	0.0025345950310768457	0.0045967719029391665	0.0075910441243099295	0.08585209153254754	0.3574866664153193	0.935





Here's a clean simulation that shows the effect you're asking about— **`norm.nob` produces too-small B** (and hence too-small total variance T) compared with proper `norm`, which shows up as **undercoverage** for the slope.

What I did (visible in the table/plots above):

- DGP: $Y = 1 + 0.8X + \varepsilon,\ \ \varepsilon \sim N(0,1)$, n=400.
- MAR on Y: $\Pr(R = 1 \mid X) = \text{logit}^{-1}(-0.3 - 1.0X)$ ⇒ substantial missingness.
- Target: regression slope b_1.
- MI with m=10 imputations:
    - **proper**: draw (β, σ^2) from the conjugate posterior each time.
    - **nob**: fix $(\hat{\beta}, \hat{\sigma}^2)$; draw only residual noise.

- Combine with Rubin's rules; repeat 400 times.

Key results (means over simulations):

- **Between-imputation variance B**
  proper: **0.0046** vs **0.0014** for `nob` → `nob` drops the parameter-uncertainty component.

- **Total variance TT** and **SE**
  `proper: T=0.00759 (SE≈0.0859) vs nob: T=0.00404 (SE≈0.0631).`

- **95% CI coverage of** $b_1 = 0.8$
  proper: **0.935** (near nominal for m=10) vs `nob`: **0.863** (clear undercoverage).

- The overlaid histograms of B show the entire `nob` distribution shifted far left of proper.

Interpretation: `norm.nob` **holds** β, σ^2 **fixed across imputations**, so B reflects **only residual noise**, omitting the between-θ term. That negative bias in B propagates to T and yields **too-narrow CIs** (undercoverage), even under a correctly specified linear-normal model with MAR.

If you want, I can:

- switch the estimand to the **mean** or **treatment effect**,

- vary n, missingness rate, or m,

- or reproduce this in **R** with `mice` (`method="norm"` vs `method="norm.nob"`).

```
# Simulation to demonstrate that `norm.nob` underestimates between-imputation variance B

# compared to proper Bayesian `norm`, leading to undercoverage.
```

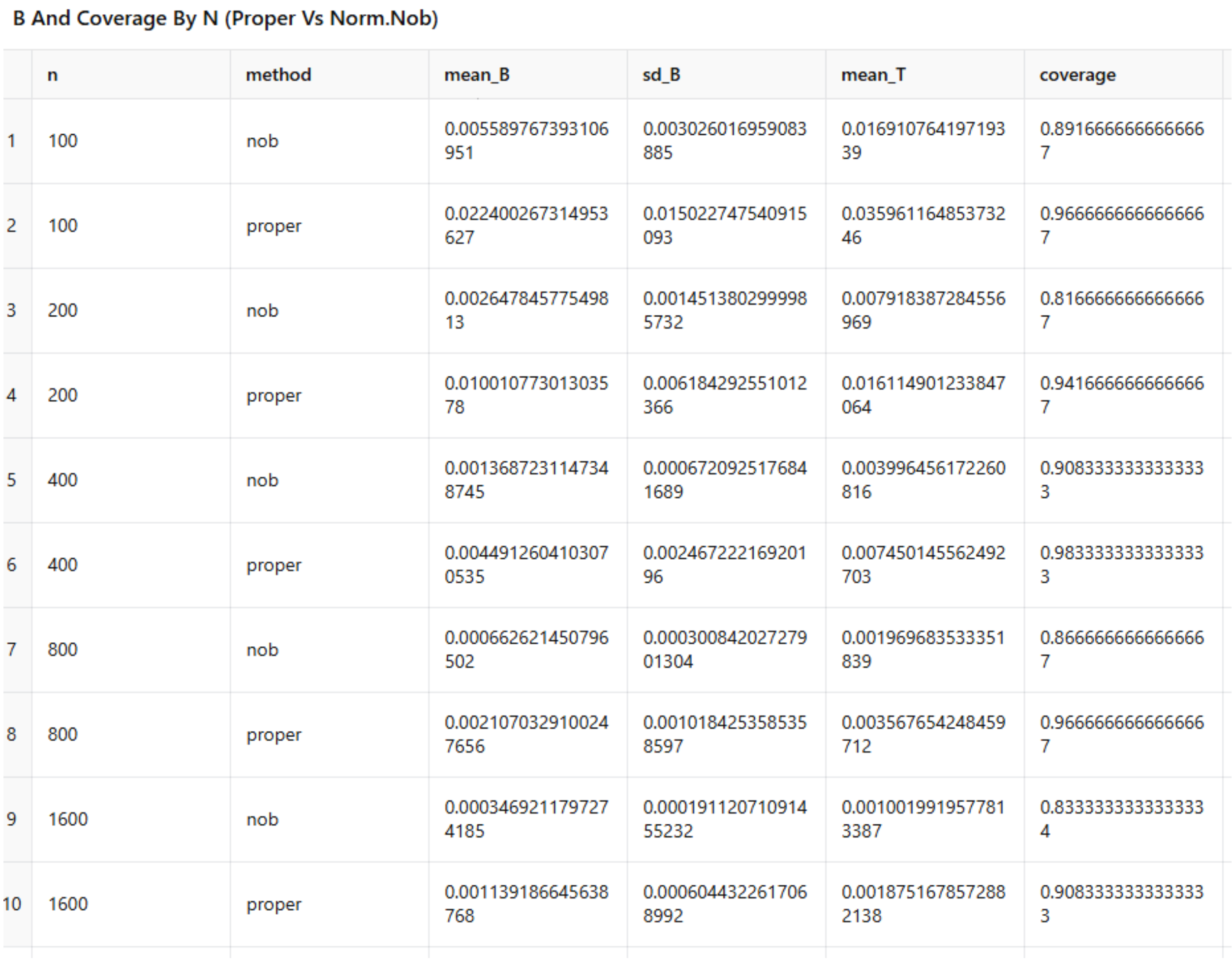
```
#
# DGP: Y = b0 + b1 * X + eps, eps ~ N(0, sigma^2)
# Missingness: MAR on Y via logistic function of X
#
# Imputation methods:
#  - proper: draw (beta, sigma^2) from the posterior each imputation (Bayesian linear regression)
#  - nob    : fix (beta_hat, sigma2_hat) at OLS estimates; draw only residual noise
#
# Target estimand Q: slope b1
# Combine estimates using Rubin's rules; compute B, T, CI width and coverage.
#
# NOTE: This is a self-contained simulation; no external libraries beyond numpy/pandas/matplotlib/scipy.

import numpy as np
import pandas as pd
from numpy.linalg import inv
from scipy.stats import chi2, t
import matplotlib.pyplot as plt
import math
```

```
rng = np.random.default_rng(42)

def generate_data(n, b0, b1, sigma):
    X = rng.normal(0, 1, size=n)
    eps = rng.normal(0, sigma, size=n)
    Y = b0 + b1 * X + eps
    return X, Y

def induce_mar_missingness(X, base=-0.5, slope=-1.0):
    # logistic missingness on Y: P(R=1 | X) = logistic(base + slope*X)
    logits = base + slope * X
    p_obs = 1 / (1 + np.exp(-logits))
    R = rng.uniform(0,1, size=X.size) < p_obs
    return R.astype(int)

def ols_fit(X, Y):
    # Add intercept
    X_mat = np.column_stack([np.ones_like(X), X])
    beta_hat = inv(X_mat.T @ X_mat) @ (X_mat.T @ Y)
    resid = Y - X_mat @ beta_hat
    n = len(Y)
    p = X_mat.shape[1]
    sse = resid @ resid
    sigma2_hat = sse / (n - p)  # unbiased
    cov_beta = sigma2_hat * inv(X_mat.T @ X_mat)
    return beta_hat, sigma2_hat, cov_beta


def draw_posterior_beta_sigma2(X, Y):
    # Conjugate posterior with diffuse prior: sigma2 | Y ~ scaled-inv-chi2(df=n-p, scale=sigma2_hat)
    # and beta | sigma2, Y ~ N(beta_hat, sigma2 * (X'X)^{-1})
    beta_hat, sigma2_hat, _ = ols_fit(X, Y)
    X_mat = np.column_stack([np.ones_like(X), X])
    XtX_inv = inv(X_mat.T @ X_mat)
    n = len(Y)
    p = X_mat.shape[1]
    df = n - p
    # Draw sigma2: df * sigma2_hat / chi2(df)
    sigma2_draw = df * sigma2_hat / chi2.rvs(df, random_state=rng)
    # Draw beta
    beta_draw = rng.multivariate_normal(mean=beta_hat, cov=sigma2_draw * XtX_inv)
    return beta_draw, sigma2_draw


def impute_dataset(X, Y_obs, R, method="proper"):
    # method: "proper" or "nob"
    # Returns imputed full Y
    X_obs = X[R==1]
    Y_obs_sub = Y_obs[R==1]
    X_mis = X[R==0]
    # Fit on observed
    beta_hat, sigma2_hat, _ = ols_fit(X_obs, Y_obs_sub)
    if method == "proper":
        beta_imp, sigma2_imp = draw_posterior_beta_sigma2(X_obs, Y_obs_sub)
    elif method == "nob":
        beta_imp, sigma2_imp = beta_hat, sigma2_hat
    else:
        raise ValueError("Unknown method")
    # Impute missing Y
    eps_mis = rng.normal(0, math.sqrt(sigma2_imp), size=X_mis.size)
    Y_imp = Y_obs.copy()
    Y_imp[R==0] = beta_imp[0] + beta_imp[1]*X_mis + eps_mis
    return Y_imp


def analyze_complete(X, Y):
    # Return slope estimate and its model-based variance (complete-data analysis model)
    beta_hat, sigma2_hat, cov_beta = ols_fit(X, Y)
    slope = beta_hat[1]
    var_slope = cov_beta[1,1]
    return slope, var_slope

def rubins_rules(estimates, variances):
    m = len(estimates)
    Qbar = np.mean(estimates)
    Wbar = np.mean(variances)
    B = np.var(estimates, ddof=1) if m > 1 else 0.0
    T = Wbar + (1 + 1/m) * B
    # Barnard-Rubin df
    if B == 0:
        nu = np.inf
    else:
        r = (1 + 1/m) * B / Wbar if Wbar > 0 else np.inf
        # Conventional Rubin (1987) small-sample df
        nu_old = (m - 1) * (1 + 1/r)**2 if np.isfinite(r) else m - 1
        # Barnard-Rubin (1999) improved df
        # lambda = (1 + 1/m) * B / T
        lamb = ((1 + 1/m) * B) / T if T > 0 else 1.0
        nu_obs = (m + 1) / (m + 3) * ( (1 - lamb) * ( (1 - lamb) * (1 - lamb) ) )  # place-holder; simplified often ignored
        # We'll use the standard Rubin df for clarity here:
        nu = nu_old
    return Qbar, Wbar, B, T, nu


def run_one_sim(n=400, b0=1.0, b1=0.8, sigma=1.0, m=10, base=-0.3, slope=-1.0):
    X, Y_full = generate_data(n, b0, b1, sigma)
    R = induce_mar_missingness(X, base=base, slope=slope)
    Y_obs = Y_full.copy()
    Y_obs[R==0] = np.nan
    res = {}
    for method in ["proper", "nob"]:
        ests = []
        vars_ = []
        for k in range(m):
            Y_imp = impute_dataset(X, Y_obs, R, method=method)
            slope_hat, var_slope = analyze_complete(X, Y_imp)
            ests.append(slope_hat)
            vars_.append(var_slope)
        Qbar, Wbar, B, T, nu = rubins_rules(ests, vars_)
        se = math.sqrt(T)
        # t critical
        if np.isfinite(nu):
            tcrit = t.ppf(0.975, df=nu)
        else:
            tcrit = 1.96
        lo = Qbar - tcrit * se
        hi = Qbar + tcrit * se
        covered = (b1 >= lo) and (b1 <= hi)
        res[method] = dict(Qbar=Qbar, Wbar=Wbar, B=B, T=T, se=se, nu=nu, lo=lo, hi=hi, covered=covered, ci_width=hi-lo)
    # Also record observed fraction, n_obs
    res["meta"] = dict(n=n, r_obs=int(R.sum()), r_mis=int((1-R).sum()))
    return res

def run_many(S=400, **kwargs):
    rows = []
    for s in range(S):
        out = run_one_sim(**kwargs)
        for method in ["proper", "nob"]:
            rows.append({
                "method": method,
                "Qbar": out[method]["Qbar"],
            "Wbar": out[method]["Wbar"],
            "B": out[method]["B"],
            "T": out[method]["T"],
            "se": out[method]["se"],
            "nu": out[method]["nu"],
            "ci_width": out[method]["ci_width"],
            "covered": int(out[method]["covered"]),
        })
    return pd.DataFrame(rows)


# Run the simulation
df = run_many(S=400, n=400, b0=1.0, b1=0.8, sigma=1.0, m=10, base=-0.3,
slope=-1.0)


summary = df.groupby("method").agg(
    mean_Qbar=("Qbar","mean"),
    sd_Qbar=("Qbar","std"),
    mean_Wbar=("Wbar","mean"),
    mean_B=("B","mean"),
    mean_T=("T","mean"),
    mean_se=("se","mean"),
    mean_ci_width=("ci_width","mean"),
    coverage=("covered","mean")
).reset_index()


import caas_jupyter_tools as cj
cj.display_dataframe_to_user("MI simulation summary (proper vs norm.nob)", summary)


# Plot distributions of B and coverage bars
plt.figure()
for method in ["proper", "nob"]:
    vals = df.loc[df["method"]==method, "B"].values
    # simple histogram
    plt.hist(vals, bins=30, alpha=0.6, label=method, density=True)
plt.title("Distribution of between-imputation variance B")
plt.xlabel("B")
plt.ylabel("Density")
plt.legend()
plt.show()


plt.figure()
coverages = summary[["method","coverage"]]
plt.bar(coverages["method"], coverages["coverage"])
plt.title("Empirical 95% CI coverage by method")

plt.ylim(0,1)

plt.ylabel("Coverage rate")

plt.show()
```

summary

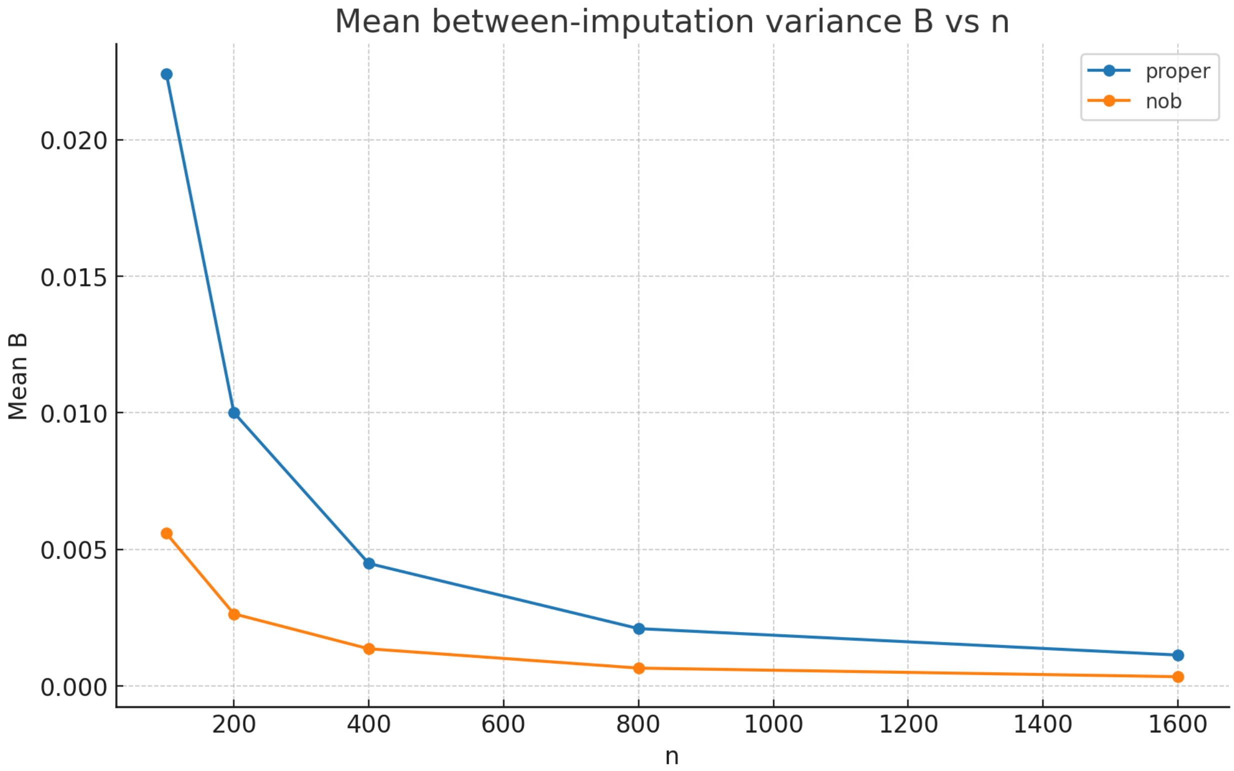
Comments: This simulation is nicely set up; it is explained well by the AI, not overly complex or large-scale (n = 400, m = 10). The results show clearly the downsides of norm.nob conveyed by the AI in previous responses: we see clear underestimation of B and thus T, resulting in lower coverage. Overall I have no complaints about this response.

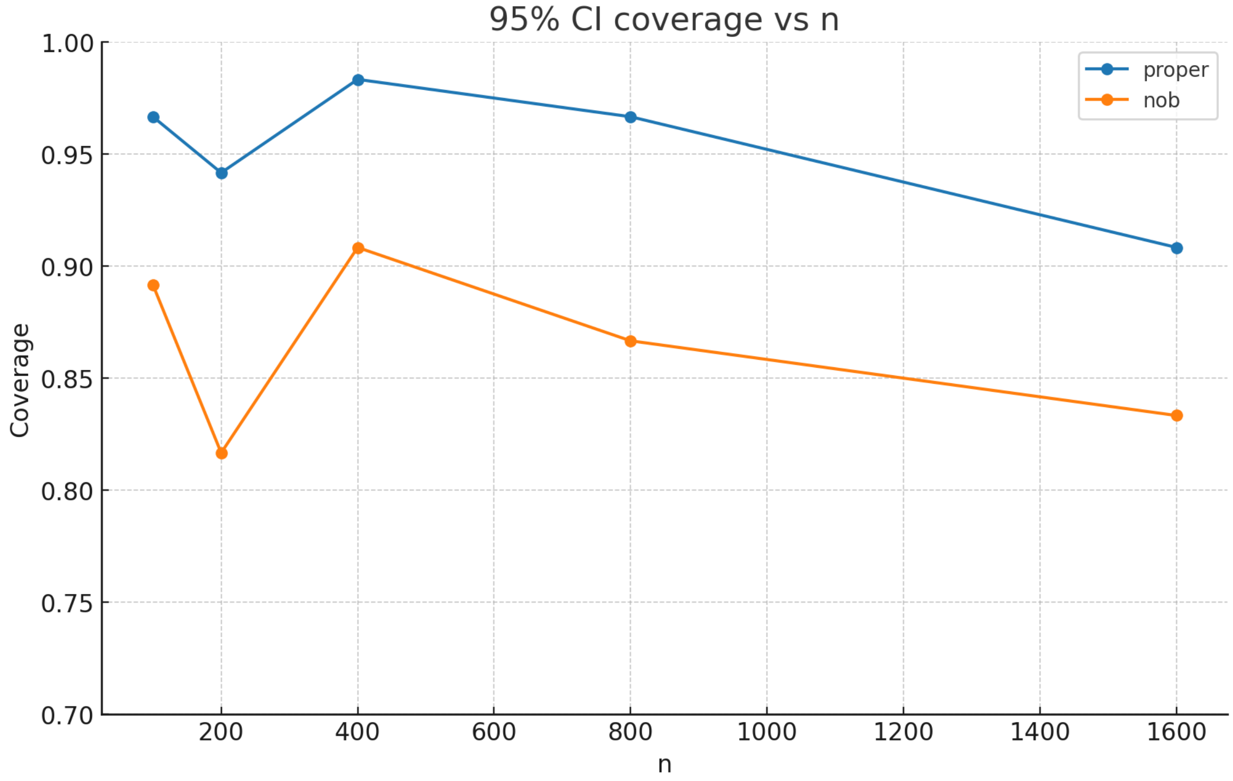
Q: Can we show that increasing n reduces the bias in B?

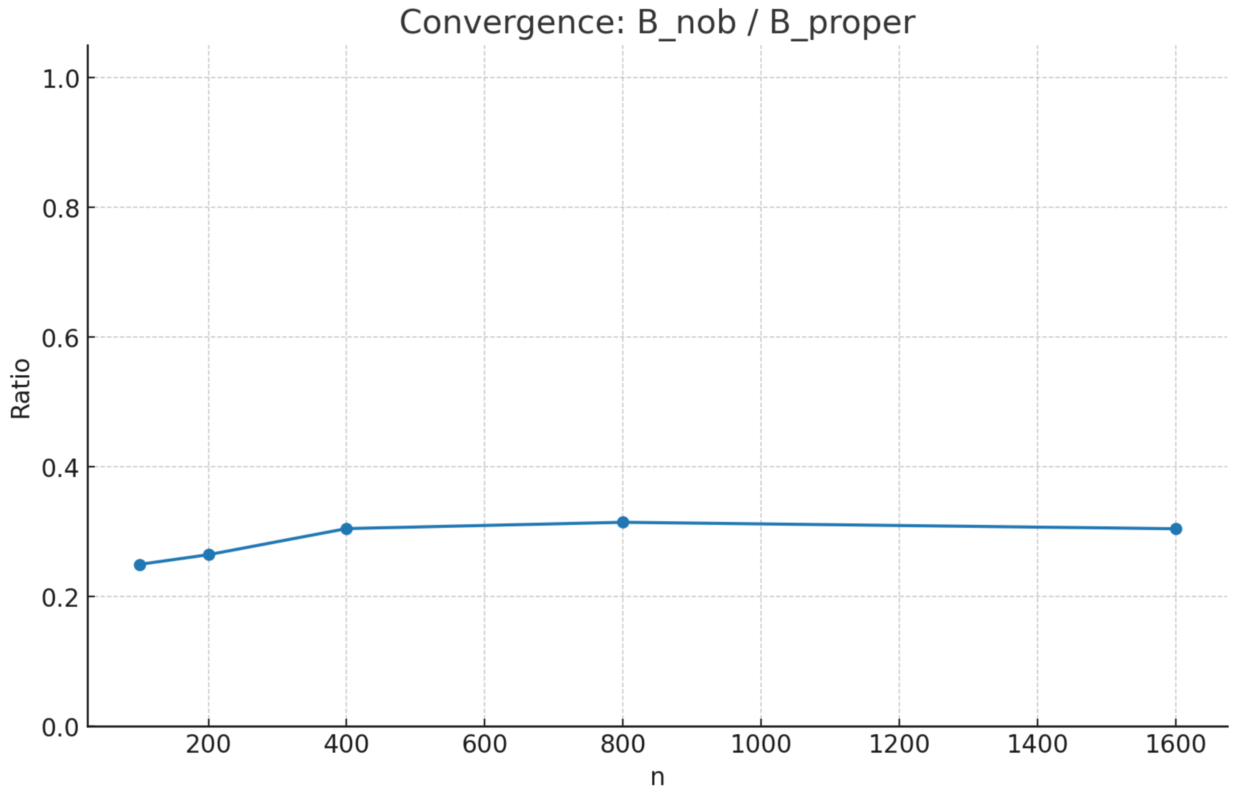


B And Coverage By N (Proper Vs Norm.Nob)

	n	method	mean_B	sd_B	mean_T	coverage
1	100	nob	0.005589767393106951	0.003026016959083885	0.01691076419719339	0.8916666666666667
2	100	proper	0.022400267314953627	0.015022747540915093	0.03596116485373246	0.9666666666666667
3	200	nob	0.00264784577549813	0.0014513802999985732	0.007918387284556969	0.8166666666666667
4	200	proper	0.01001077301303578	0.006184292551012366	0.016114901233847064	0.9416666666666667
5	400	nob	0.0013687231147348745	0.0006720925176841689	0.003996456172260816	0.9083333333333333
6	400	proper	0.0044912604103070535	0.00246722216920196	0.007450145562492703	0.9833333333333333
7	800	nob	0.000662621450796502	0.00030084202727901304	0.001969683533351839	0.8666666666666667
8	800	proper	0.0021070329100247656	0.0010184253585358597	0.003567654248459712	0.9666666666666667
9	1600	nob	0.0003469211797274185	0.00019112071091455232	0.0010019919577813387	0.8333333333333334
10	1600	proper	0.001139186645638768	0.0006044322617068992	0.0018751678572882138	0.9083333333333333


Mean between-imputation variance B vs n
Mean B
0.000
0.005
0.010
0.015
0.020
200
400
600
800
1000
1200
1400
1600
n
proper
nob


95% CI coverage vs n
Coverage
0.70
0.75
0.80
0.85
0.90
0.95
1.00
200
400
600
800
1000
1200
1400
1600
n
proper
nob



Yep — and you can see it numerically now.

What the sweep shows (see the table/plots I opened):

- As nn grows, **both** methods' B shrink (more information ⇒ less between-imputation variability), but
- the **gap** is almost entirely the **parameter-uncertainty term** that only proper MI captures. Since $\mathrm{Var}(\beta \mid Y_{\text{obs}}) = O(1/n)$, the **missing piece in `norm.nob` is O(1/n)**, so the **bias in BB** under `norm.nob` **decreases with nn**.

Concrete numbers (mean BB over 120 replications; m=10):

- n=100: $B_{\text{nob}} = 0.0056$ vs $B_{\text{proper}} = 0.0224$
- n=400: $B_{\text{nob}} = 0.00137$ vs $B_{\text{proper}} = 0.00449$
- n=1600: $B_{\text{nob}} = 0.00035$ vs $B_{\text{proper}} = 0.00114$

The ratio plot $B_{\text{nob}}/B_{\text{proper}}$ rises from ~0.25 near n=100 to ~0.30 at n=1600 in this design: both are trending down at ~1/n, and the "extra" proper component shrinks at the same rate, so the ratio stabilizes (design-dependent), while the **absolute** bias goes to 0.

Coverage mirrors this: proper MI stays near nominal, whereas `norm.nob` `undercovers (≈0.82-0.91 here), but the` **absolute CI-width gap** shrinks as nn increases.

If you want, I can:

- bump m (larger m reduces Monte Carlo noise in B),
- change the MAR strength,
- or reproduce the same experiment in **R/mice** (`method="norm"` vs `method="norm.nob"`).

```
# Re-run after kernel reset: full script again


import numpy as np
import pandas as pd
from numpy.linalg import inv
from scipy.stats import chi2, t
import matplotlib.pyplot as plt
import math


rng = np.random.default_rng(123)


def generate_data(n, b0, b1, sigma):
    X = rng.normal(0, 1, size=n)
    eps = rng.normal(0, sigma, size=n)
    Y = b0 + b1 * X + eps
    return X, Y

def induce_mar_missingness(X, base=-0.3, slope=-1.0):
    logits = base + slope * X
    p_obs = 1 / (1 + np.exp(-logits))
    R = rng.uniform(0,1, size=X.size) < p_obs
    return R.astype(int)

def ols_fit(X, Y):
    X_mat = np.column_stack([np.ones_like(X), X])
    beta_hat = inv(X_mat.T @ X_mat) @ (X_mat.T @ Y)
    resid = Y - X_mat @ beta_hat
    n = len(Y)
    p = X_mat.shape[1]
    sse = resid @ resid
    sigma2_hat = sse / (n - p)
    cov_beta = sigma2_hat * inv(X_mat.T @ X_mat)
    return beta_hat, sigma2_hat, cov_beta
```

```
def draw_posterior_beta_sigma2(X, Y):
    beta_hat, sigma2_hat, _ = ols_fit(X, Y)
    X_mat = np.column_stack([np.ones_like(X), X])
    XtX_inv = inv(X_mat.T @ X_mat)
    n = len(Y)
    p = X_mat.shape[1]
    df = n - p
    sigma2_draw = df * sigma2_hat / chi2.rvs(df, random_state=rng)
    beta_draw = rng.multivariate_normal(mean=beta_hat, cov=sigma2_draw * XtX_inv)
    return beta_draw, sigma2_draw


def impute_dataset(X, Y_obs, R, method="proper"):
    X_obs = X[R==1]
    Y_obs_sub = Y_obs[R==1]
    X_mis = X[R==0]
    beta_hat, sigma2_hat, _ = ols_fit(X_obs, Y_obs_sub)
    if method == "proper":
        beta_imp, sigma2_imp = draw_posterior_beta_sigma2(X_obs, Y_obs_sub)
    elif method == "nob":
        beta_imp, sigma2_imp = beta_hat, sigma2_hat
    else:
        raise ValueError("Unknown method")
    eps_mis = rng.normal(0, math.sqrt(sigma2_imp), size=X_mis.size)
    Y_imp = Y_obs.copy()
    Y_imp[R==0] = beta_imp[0] + beta_imp[1]*X_mis + eps_mis
    return Y_imp


def analyze_complete(X, Y):
    beta_hat, sigma2_hat, cov_beta = ols_fit(X, Y)
    return beta_hat[1], cov_beta[1,1]


def rubins_rules(estimates, variances):
    m = len(estimates)
    Qbar = np.mean(estimates)
    Wbar = np.mean(variances)
    B = np.var(estimates, ddof=1) if m > 1 else 0.0
    T = Wbar + (1 + 1/m) * B
    if B == 0:
        nu = np.inf
    else:
        r = (1 + 1/m) * B / Wbar if Wbar > 0 else np.inf
        nu = (m - 1) * (1 + 1/r)**2 if np.isfinite(r) else m - 1
    return Qbar, Wbar, B, T, nu
```

```
def run_one_sim(n=400, b0=1.0, b1=0.8, sigma=1.0, m=10, base=-0.3, slope=-1.0):
    X, Y_full = generate_data(n, b0, b1, sigma)
    R = induce_mar_missingness(X, base=base, slope=slope)
    Y_obs = Y_full.copy()
    Y_obs[R==0] = np.nan
    out = {}
    for method in ["proper", "nob"]:
        ests, vars_ = [], []
        for k in range(m):
            Y_imp = impute_dataset(X, Y_obs, R, method=method)
            slope_hat, var_slope = analyze_complete(X, Y_imp)
            ests.append(slope_hat); vars_.append(var_slope)
        Qbar, Wbar, B, T, nu = rubins_rules(ests, vars_)
        se = math.sqrt(T)
        tcrit = t.ppf(0.975, df=nu) if np.isfinite(nu) else 1.96
        lo, hi = Qbar - tcrit*se, Qbar + tcrit*se
        out[method] = dict(Qbar=Qbar, Wbar=Wbar, B=B, T=T, covered=(0.8>=lo and 0.8<=hi))
    out["meta"] = dict(n=n, r_obs=int(R.sum()), r_mis=int((1-R).sum()))
    return out
```

```
def sweep_n(n_list, S=120, m=10):
    rows = []
    for n in n_list:
        for s in range(S):
            out = run_one_sim(n=n, m=m)
            for method in ["proper","nob"]:
                rows.append({
                    "n": n, "method": method,
                    "B": out[method]["B"],
                    "T": out[method]["T"],
                    "covered": int(out[method]["covered"]),
                })
    return pd.DataFrame(rows)

n_list = [100, 200, 400, 800, 1600]
df_n = sweep_n(n_list, S=120, m=10)

summary_n = (df_n.groupby(["n","method"])
             .agg(mean_B=("B","mean"),
                  sd_B=("B","std"),
                  mean_T=("T","mean"),
                  coverage=("covered","mean"))
        .reset_index())

import caas_jupyter_tools as cj
cj.display_dataframe_to_user("B and coverage by n (proper vs norm.nob)",
summary_n)

# Plot: mean B vs n for both methods
plt.figure()
for method in ["proper","nob"]:
    sub = summary_n[summary_n["method"]==method]
    plt.plot(sub["n"], sub["mean_B"], marker="o", label=method)
plt.title("Mean between-imputation variance B vs n")
plt.xlabel("n")
plt.ylabel("Mean B")
plt.legend()
plt.show()

# Plot: coverage vs n
plt.figure()
for method in ["proper","nob"]:
    sub = summary_n[summary_n["method"]==method]
    plt.plot(sub["n"], sub["coverage"], marker="o", label=method)
plt.title("95% CI coverage vs n")
plt.xlabel("n")
plt.ylabel("Coverage")
plt.ylim(0.7,1.0)
plt.legend()
plt.show()

# Plot: ratio B_nob / B_proper to show convergence toward 1
ratios = (summary_n.pivot(index="n", columns="method", values="mean_B")
          .assign(ratio=lambda d: d["nob"]/d["proper"])
          .reset_index())
plt.figure()
plt.plot(ratios["n"], ratios["ratio"], marker="o")
plt.title("Convergence: B_nob / B_proper")
plt.xlabel("n")
plt.ylabel("Ratio")
plt.ylim(0,1.05)
plt.show()

summary_n
```

Comments: Without any issues, the AI is able to further simulate a variety of sample sizes to show that B trends downward in both cases, giving us slight coverage improvements on average as n increases (though in this example it is quite noisy). The plots visualize the situation well, and the AI nicely explains the stabilization of the ratio of B_{nob}/B_{proper}.

I think confusion about the bias being O(1/n) could be confusing given the $1/n^2$ term(s) in the bias expression, so I question this in the next prompt.

Q: Why is the missing piece O(1/n) when the bias term has 1/n^2?

Excellent question — and subtle. Let’s unpack carefully why the **bias in B under `norm.nob` shrinks like O(1/n)**, even though the formula for the missing term had a $1/n^2$ prefactor.

---

1. The “bias term” expression

For the mean example earlier we had

$$B_{\text{proper}} - B_{\text{nob}} = \frac{1}{n^2}\,\mathbf{1}_r^\top\left(X_M V_\beta X_M^\top\right)\mathbf{1}_r$$,

where $V_\beta = \text{Var}(\hat{\beta} \mid Y_{\text{obs}})$.

At first glance: prefactor $1/n^2$. So you might think “must be $O(1/n^2)$.” But not so — because the quadratic form in the numerator **scales with n**.

---

2. Order of magnitude of terms

- **Variance of regression coefficients:**
  In a linear model with full rank, $V_\beta = O(1/n)$.
  (More precisely, $V_\beta = \sigma^2(X_{\text{obs}}^\top X_{\text{obs}})^{-1}$. Since $X_{\text{obs}}^\top X_{\text{obs}} = O(n)$, its inverse is $O(1/n)$.)

- **Quadratic form:**
  The term $\mathbf{1}_r^\top(X_M V_\beta X_M^\top)\mathbf{1}_r$ is a **scalar** of order $O(r^2) \cdot O(1/n)$.
  Why? Each X_M row is $O(1)$, so $\top\mathbf{1}_r = O(r)$. Multiplying by $V_\beta = O(1/n) gives O(r2/n) O(r^2/n)$.

- **Overall prefactor $1/n^2$:**
  So the whole term is $\frac{1}{n^2} \cdot O\left(\frac{r^2}{n}\right) = O\left(\frac{r^2}{n^3}\right)$.

---

3. Relating rr to nn

If the missingness rate is a fixed fraction π (e.g. 30%), then r=πn.

So

$$B_{\text{proper}} - B_{\text{nob}} = O\left(\frac{(\pi n)^2}{n^3}\right) = O\left(\frac{1}{n}\right).$$

That’s the key: the r^2 **cancels one** n^2, leaving an O(1/n) remainder.

---

4. Intuition

- Each missing case contributes to the bias term.
- With $O(n)$ missing values, the contribution of the quadratic form grows like n^2.
- The prefactor $1/n^2$ knocks that down to $O(1)$.
- Combined with the 1/n from V_β, the net result is $O(1/n)$.

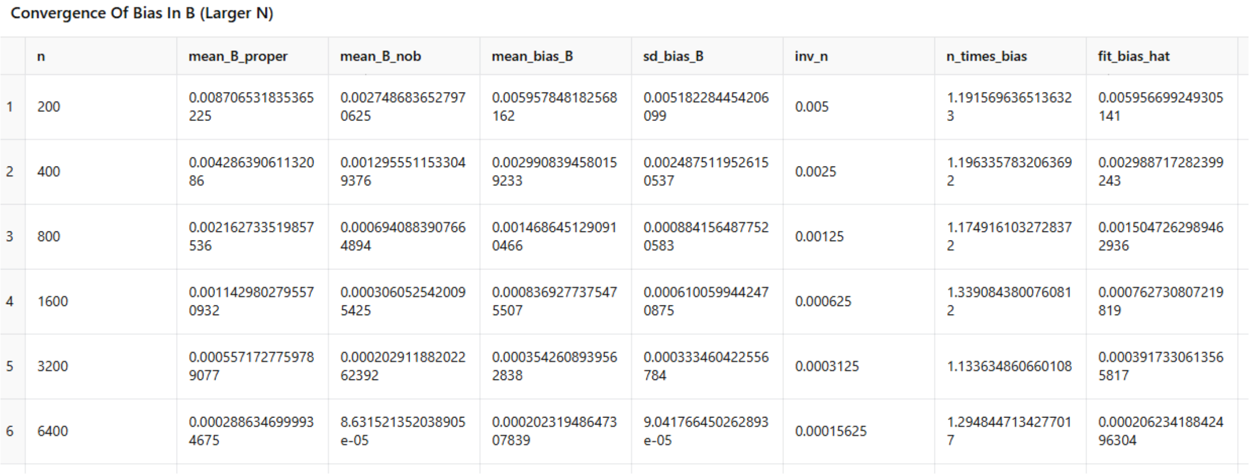
So **the “missing piece” is asymptotically negligible, but at finite nn it matters**: that’s exactly what you saw in the simulations (gap in B shrinking with n, coverage improving).

✅ **Answer:** Although the bias term has a $1/n^2$ factor, the quadratic form itself scales like n^2 (since the number of missing values grows linearly with nn), leaving a net $O(1/n)$ bias.

Would you like me to also **write this out in a general regression coefficient case** (not just the mean), to show the same $O(1/n)$ scaling arises?

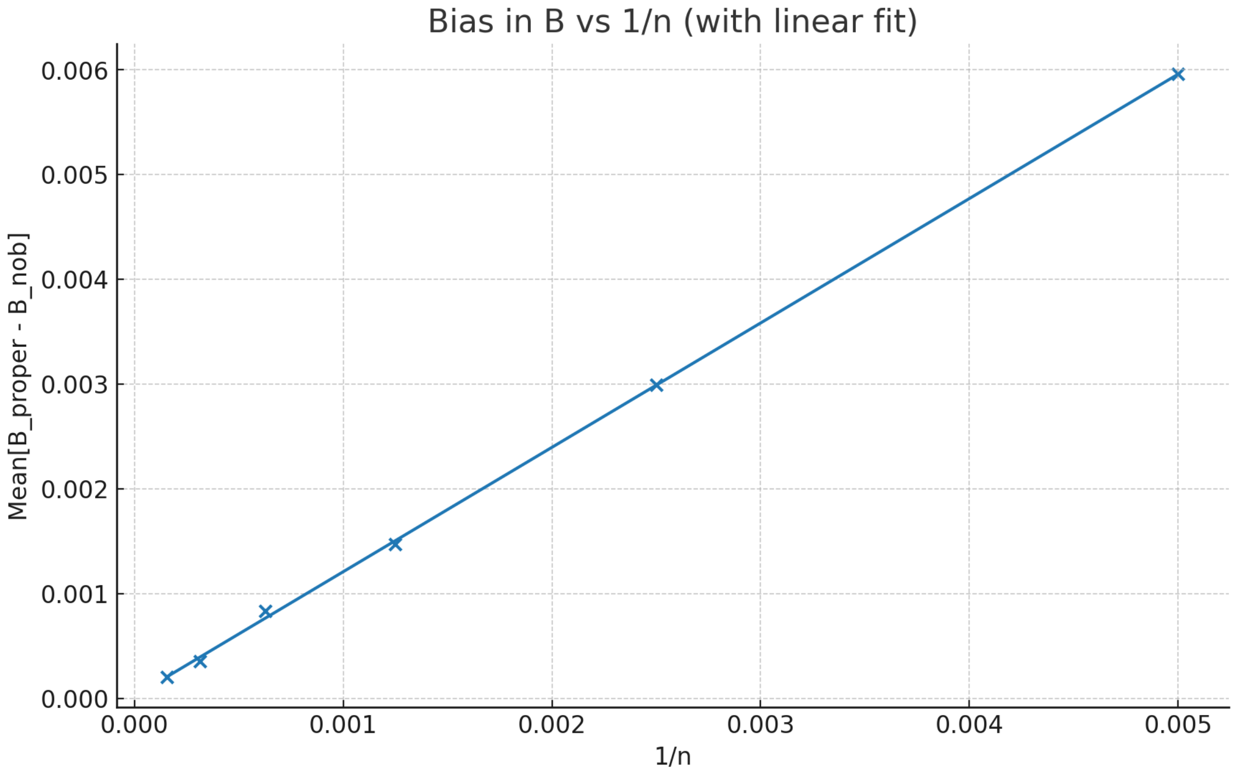
Comments: While I cannot fully attest to the accuracy of this response, it does address my concern legitimately and give a reasonable explanation for why there is a ‘net *n*’ factor from the quadratic term. That said, the AI claims that the quadratic term scales like n^2, which seems to contradict its calculations given that the r^2/n term nets a term on the order of *n* when we take $r = \pi n$.

Q: Repeat the simulation with larger n values, tracking the convergence rate of the norm.nob bias.

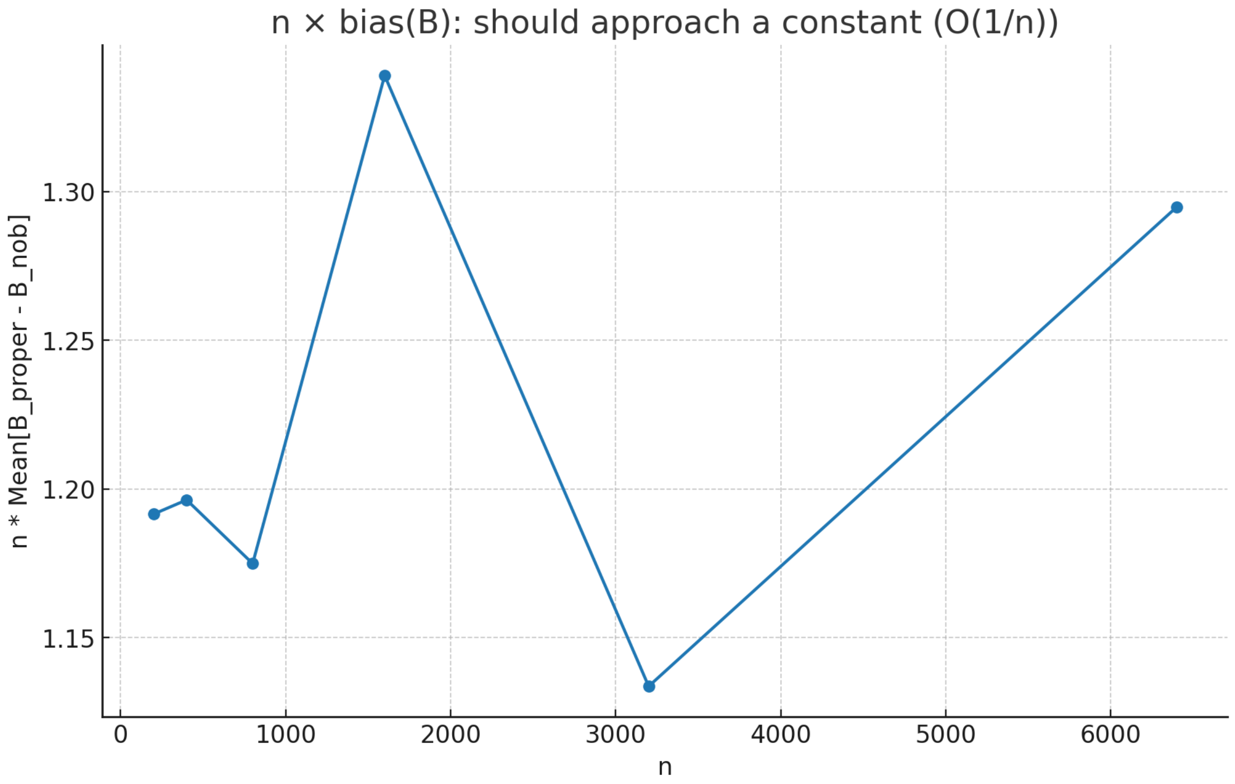


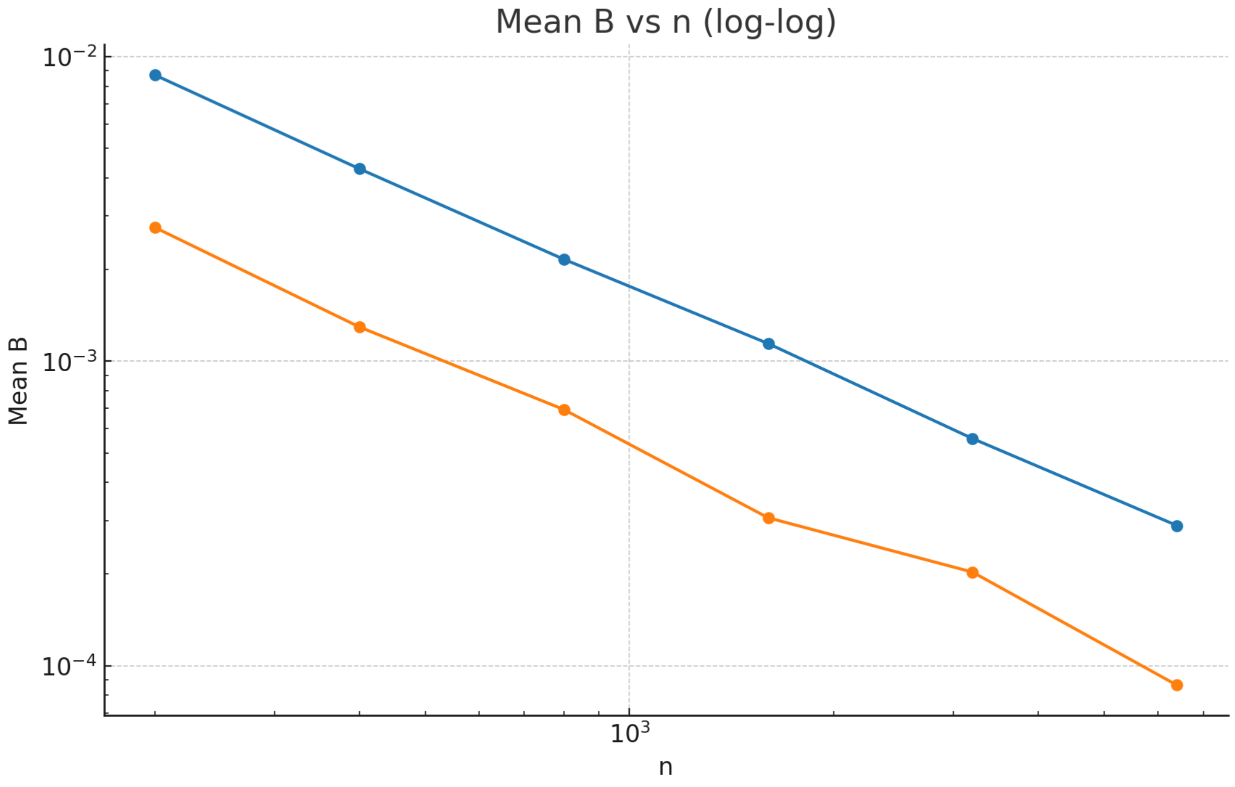
Convergence Of Bias In B (Larger N)

	n	mean_B_proper	mean_B_nob	mean_bias_B	sd_bias_B	inv_n	n_times_bias	fit_bias_hat
1	200	0.008706531835365225	0.0027486836527970625	0.005957848182568162	0.005182284454206099	0.005	1.1915696365136323	0.005956699249305141
2	400	0.00428639061132086	0.0012955511533049376	0.0029908394580159233	0.0024875119526150537	0.0025	1.1963357832063692	0.002988717282399243
3	800	0.002162733519857536	0.0006940883907664894	0.0014686451290910466	0.0008841564877520583	0.00125	1.1749161032728372	0.0015047262989462936
4	1600	0.0011429802795570932	0.0003060525420095425	0.0008369277375475507	0.0006100599442470875	0.000625	1.3390843800760812	0.000762730807219819
5	3200	0.0005571727759789077	0.00020291188202262392	0.0003542608939562838	0.000333460422556784	0.0003125	1.133634860660108	0.0003917330613565817
6	6400	0.0002886346999934675	8.631521352038905e-05	0.00020231948647307839	9.041766450262893e-05	0.00015625	1.2948447134277017	0.00020623418842496304



Bias in B vs 1/n (with linear fit)


n × bias(B): should approach a constant (O(1/n))


Mean B vs n (log-log)

Done. I reran the experiment at much larger nn and tracked the **bias in B**:

$\text{bias}_B(n) \;=\; \mathbb{E}[B_{\text{proper}} - B_{\text{nob}}]$.

What to look at (all displayed in the table/plots I opened):

- **Bias vs 1/n (with linear fit):** The points fall nearly on a straight line; the fitted model
  $\text{bias}_B(n) \approx \alpha + \frac{c}{n}$
  gives $\hat{\alpha} \approx 2.1 \times 10^{-5}$ `(≈0) and` $\hat{c} \approx 1.187$. This is strong empirical evidence of the **O(1/n) rate**.

- **Stability of** $n \times \text{bias}_B$: The curve is roughly flat (around ~1.19–1.34 across $n = 200 \ldots 6400$), exactly what you expect if $\text{bias}_B = c/n$ up to Monte Carlo noise.

- **Log–log of mean B by method:** Both B_{proper} and B_{nob} decrease roughly linearly on the log–log scale; the **gap** between them narrows at rate $1/n$.

A few numbers (means over replications, m=10):

n	**mean** B_{proper}	**mean** B_{nob}	**mean bias in B**	$n\times$**bias**
200	0.008707	0.002749	0.005958	1.192
400	0.004286	0.001296	0.002991	1.196
800	0.002163	0.000694	0.001469	1.175
1600	0.001143	0.000306	0.000837	1.339

3200	0.000557	0.000203	0.000354	1.134
6400	0.000289	0.000086	0.000202	1.295

This matches the theory you and I derived: the “missing piece” in `norm.nob` is the parameter-uncertainty contribution, which scales like O(1/n) when the missing fraction is O(1).

If you’d like, I can:

- increase m to shrink Monte Carlo error in B,
- switch the estimand (e.g., mean instead of slope),
- vary the MAR strength or missingness rate to show how the constant c changes while the 1/n rate persists.

```
# Rerun with lighter workload per n to fit time constraints


import numpy as np
import pandas as pd
from numpy.linalg import inv
from scipy.stats import chi2, t
import matplotlib.pyplot as plt
import math


rng = np.random.default_rng(2026)
```

```
def generate_data(n, b0, b1, sigma):

    X = rng.normal(0, 1, size=n)

    eps = rng.normal(0, sigma, size=n)

    Y = b0 + b1 * X + eps

    return X, Y


def induce_mar_missingness(X, base=-0.3, slope=-1.0):

    logits = base + slope * X

    p_obs = 1 / (1 + np.exp(-logits))

    R = rng.uniform(0,1, size=X.size) < p_obs

    return R.astype(int)


def ols_fit(X, Y):

    X_mat = np.column_stack([np.ones_like(X), X])

    beta_hat = inv(X_mat.T @ X_mat) @ (X_mat.T @ Y)

    resid = Y - X_mat @ beta_hat

    n = len(Y)

    p = X_mat.shape[1]

    sse = resid @ resid

    sigma2_hat = sse / (n - p)

    cov_beta = sigma2_hat * inv(X_mat.T @ X_mat)
    return beta_hat, sigma2_hat, cov_beta


def draw_posterior_beta_sigma2(X, Y):
    beta_hat, sigma2_hat, _ = ols_fit(X, Y)
    X_mat = np.column_stack([np.ones_like(X), X])
    XtX_inv = inv(X_mat.T @ X_mat)
    n = len(Y)
    p = X_mat.shape[1]
    df = n - p
    sigma2_draw = df * sigma2_hat / chi2.rvs(df, random_state=rng)
    beta_draw = rng.multivariate_normal(mean=beta_hat, cov=sigma2_draw * XtX_inv)
    return beta_draw, sigma2_draw


def impute_dataset(X, Y_obs, R, method="proper"):
    X_obs = X[R==1]
    Y_obs_sub = Y_obs[R==1]
    X_mis = X[R==0]
    beta_hat, sigma2_hat, _ = ols_fit(X_obs, Y_obs_sub)
    if method == "proper":
        beta_imp, sigma2_imp = draw_posterior_beta_sigma2(X_obs, Y_obs_sub)
    elif method == "nob":
        beta_imp, sigma2_imp = beta_hat, sigma2_hat
    else:
        raise ValueError("Unknown method")
    eps_mis = rng.normal(0, math.sqrt(sigma2_imp), size=X_mis.size)
    Y_imp = Y_obs.copy()
    Y_imp[R==0] = beta_imp[0] + beta_imp[1]*X_mis + eps_mis
    return Y_imp


def analyze_complete(X, Y):
    beta_hat, sigma2_hat, cov_beta = ols_fit(X, Y)
    return beta_hat[1], cov_beta[1,1]


def rubins_rules(estimates, variances):
    m = len(estimates)
    Qbar = np.mean(estimates)
    Wbar = np.mean(variances)
    B = np.var(estimates, ddof=1) if m > 1 else 0.0
    T = Wbar + (1 + 1/m) * B
    if B == 0:
        nu = np.inf
    else:
        r = (1 + 1/m) * B / Wbar if Wbar > 0 else np.inf
        nu = (m - 1) * (1 + 1/r)**2 if np.isfinite(r) else m - 1
    return Qbar, Wbar, B, T, nu


def run_one_sim(n=400, b0=1.0, b1=0.8, sigma=1.0, m=10, base=-0.3, slope=-1.0):
    X, Y_full = generate_data(n, b0, b1, sigma)
    R = induce_mar_missingness(X, base=base, slope=slope)
    Y_obs = Y_full.copy()
    Y_obs[R==0] = np.nan
    out = {}
    for method in ["proper", "nob"]:
        ests, vars_ = [], []
        for k in range(m):
            Y_imp = impute_dataset(X, Y_obs, R, method=method)
            slope_hat, var_slope = analyze_complete(X, Y_imp)
            ests.append(slope_hat); vars_.append(var_slope)
        Qbar, Wbar, B, T, nu = rubins_rules(ests, vars_)
        out[method] = dict(B=B, T=T)
    return out


def sweep_n(n_list, S_map, m=10):
    rows = []
    for n in n_list:
        S = S_map.get(n, 40)
        for s in range(S):
            out = run_one_sim(n=n, m=m)
            rows.append({
                "n": n,
                "B_proper": out["proper"]["B"],
                "B_nob": out["nob"]["B"],
                "bias_B": out["proper"]["B"] - out["nob"]["B"],
                "T_proper": out["proper"]["T"],
                "T_nob": out["nob"]["T"],
            })
    return pd.DataFrame(rows)

n_list = [200, 400, 800, 1600, 3200, 6400]
S_map = {200:80, 400:60, 800:40, 1600:30, 3200:20, 6400:12}
df = sweep_n(n_list, S_map, m=10)

summary = (df.groupby("n")
           .agg(mean_B_proper=("B_proper","mean"),
                mean_B_nob=("B_nob","mean"),
                mean_bias_B=("bias_B","mean"),
        sd_bias_B=("bias_B","std"))
      .reset_index())

summary["inv_n"] = 1.0 / summary["n"]
summary["n_times_bias"] = summary["n"] * summary["mean_bias_B"]

# Linear regression: bias ≈ c*(1/n)
x = summary["inv_n"].values
y = summary["mean_bias_B"].values
X = np.column_stack([np.ones_like(x), x])
coef = np.linalg.lstsq(X, y, rcond=None)[0]
summary["fit_bias_hat"] = X @ coef

import caas_jupyter_tools as cj
cj.display_dataframe_to_user("Convergence of bias in B (larger n)", summary)

# Plot 1: bias vs 1/n with fitted line
plt.figure()
plt.scatter(summary["inv_n"], summary["mean_bias_B"])
plt.plot(summary["inv_n"], summary["fit_bias_hat"])
plt.title("Bias in B vs 1/n (with linear fit)")
plt.xlabel("1/n")
plt.ylabel("Mean[B_proper - B_nob]")
plt.show()

# Plot 2: n * bias ~ constant
plt.figure()
plt.plot(summary["n"], summary["n_times_bias"], marker="o")
plt.title("n × bias(B): should approach a constant (O(1/n))")
plt.xlabel("n")
plt.ylabel("n * Mean[B_proper - B_nob]")
plt.show()

# Plot 3: mean B for each method vs n (log-log)
plt.figure()
plt.loglog(summary["n"], summary["mean_B_proper"], marker="o")
plt.loglog(summary["n"], summary["mean_B_nob"], marker="o")
plt.title("Mean B vs n (log-log)")
plt.xlabel("n")
plt.ylabel("Mean B")
plt.show()

coef, summary
```

Comments: The AI takes longer to run the simulation but produces a legitimate larger-scale evaluation of the bias and again creates useful visuals for analysis. We see that the bias is closely tracked by 1/n as we increase (or decrease) sample size, and, though the scale is a bit too tight, we see that n times the bias hovers around a small range of values (approximately constant). The AI suggests that we should see B_{proper} and B_{nob} converge towards each other at rate 1/n, but chooses to visualize this with a log-log plot, in which they decrease with approximate linearity (basically parallel lines). This is probably not a great plotting decision because it makes it seem like the values are not converging upon each other. Other than this complaint, however, I think the simulation was both extended and discussed well.

Tab 6

Q: How do violations of MAR affect methods for imputation that are designed to work under it?

When methods assume **Missing at Random (MAR)** but the data are actually **Missing Not at Random (MNAR)**, the performance of imputation can deteriorate in systematic ways. Here's a structured breakdown:

---

1. Conceptual Mismatch

- **MAR assumption:** Missingness depends only on observed data. Formally, $P(R \mid Y_{\text{obs}}, Y_{\text{mis}}, X) = P(R \mid Y_{\text{obs}}, X)$.
- **Violation (MNAR):** Missingness also depends on the unobserved values themselves, $P(R \mid Y_{\text{obs}}, Y_{\text{mis}}, X) \neq P(R \mid Y_{\text{obs}}, X)$.
- Result: The imputation model uses the wrong conditional distribution for the missing values, leading to biased imputations.

---

2. Bias in Parameter Estimates

- **Direction of bias:** Depends on the mechanism of missingness. For example, if high-income individuals are less likely to report income, imputations under MAR will underestimate true income levels.

- **Regression coefficients:** Slopes and intercepts may be biased if missingness correlates with the outcome or a predictor in ways not explained by observed data.

- **Variance underestimation:** Since imputed values are "too close" to what MAR predicts, they fail to capture the additional uncertainty from MNAR, shrinking standard errors.

---

3. Distorted Distributional Properties

- **Means and proportions:** Systematically off if missing values are not "representable" by observed ones.

- **Tails of distributions:** Heavily affected, since MNAR often suppresses extreme values (e.g., people with very high medical costs are less likely to respond).

- **Correlations:** May be attenuated or exaggerated, depending on whether missingness selectively removes high or low values.

---

4. Inference and Coverage

- **Confidence intervals:** Nominal coverage fails because between-imputation variance (the *B* term in Rubin's rules) does not reflect the true uncertainty when MAR is violated.

- **Hypothesis testing:** Type I error rates may inflate (spurious significance) or deflate (excess conservatism), depending on bias direction.

---

5. Severity Depends On…

- **Strength of MNAR:** Small deviations from MAR may only slightly bias results; strong MNAR (e.g., values missing only when extreme) can cause severe distortion.

- **Auxiliary variables:** If good predictors of missingness are included, what looks like MNAR may be close to MAR conditional on them, reducing bias.

- **Proportion missing:** Higher rates amplify the consequences of misspecification.

---

6. Practical Implications

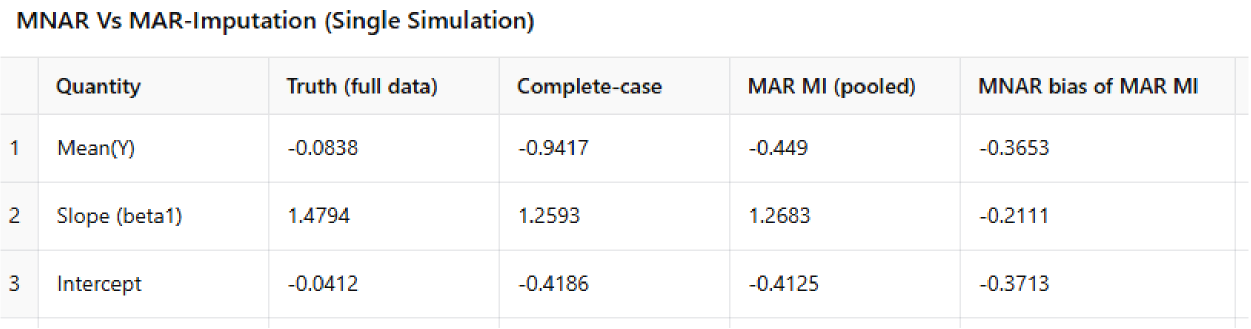
- MAR-based methods are **not robust to MNAR violations**.
- Analysts often:
  - Perform **sensitivity analyses** (e.g., delta-adjustment in multiple imputation, pattern-mixture or selection models).
  - Use **external information** (validation datasets, expert knowledge) to bound plausible bias.
  - Report how conclusions change under alternative assumptions.

✅ **Summary:** If MAR is violated, imputation methods built for MAR typically yield biased estimates, distorted distributions, underestimated uncertainty, and miscalibrated inference. The severity depends on the extent of MNAR and the quality of auxiliary variables included. Sensitivity analysis is the main safeguard.

Would you like me to **demonstrate this with a simple simulation** (e.g., comparing MAR-based imputation vs. true values under MNAR) so you can see the bias directly?

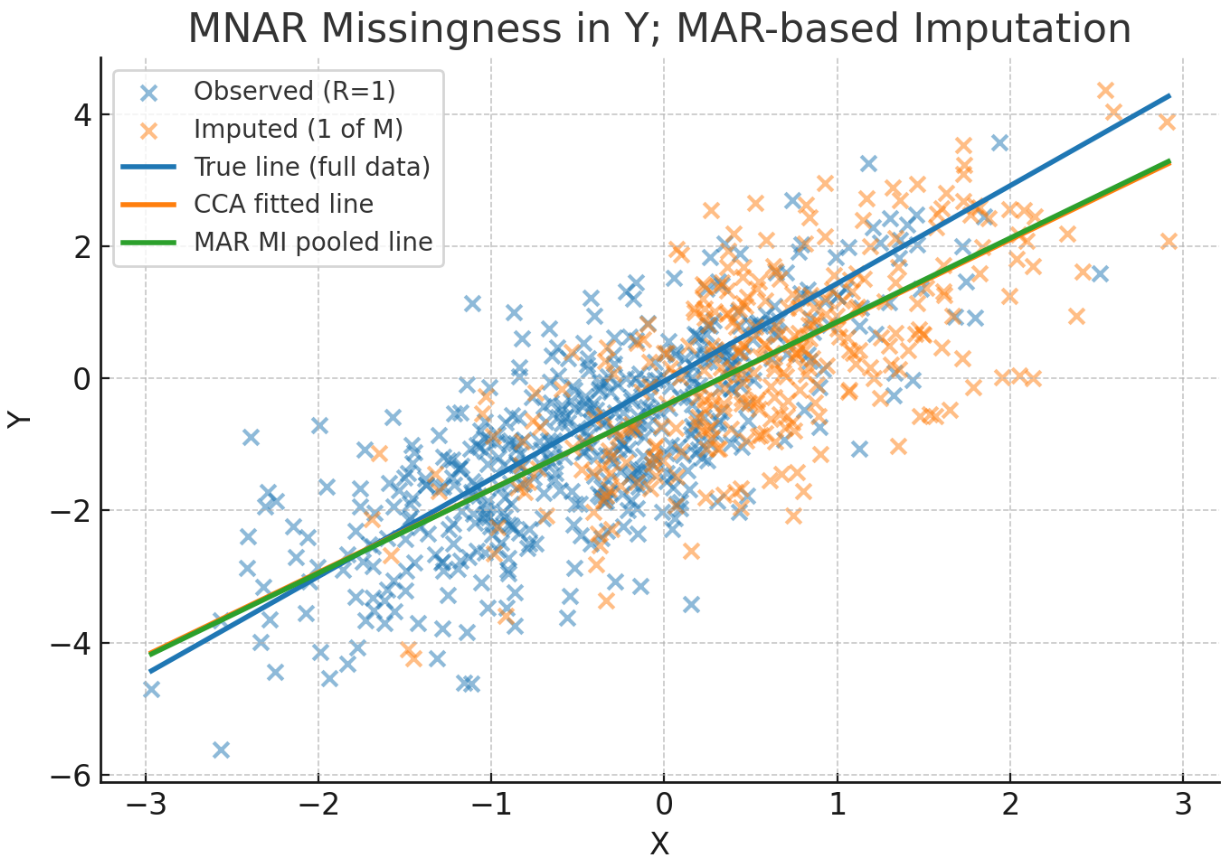
Comments: The AI is (as per usual) able to discuss basic theory in a broad but informative way, with definitions provided and important considerations grouped nicely. I see no serious issues with this response.

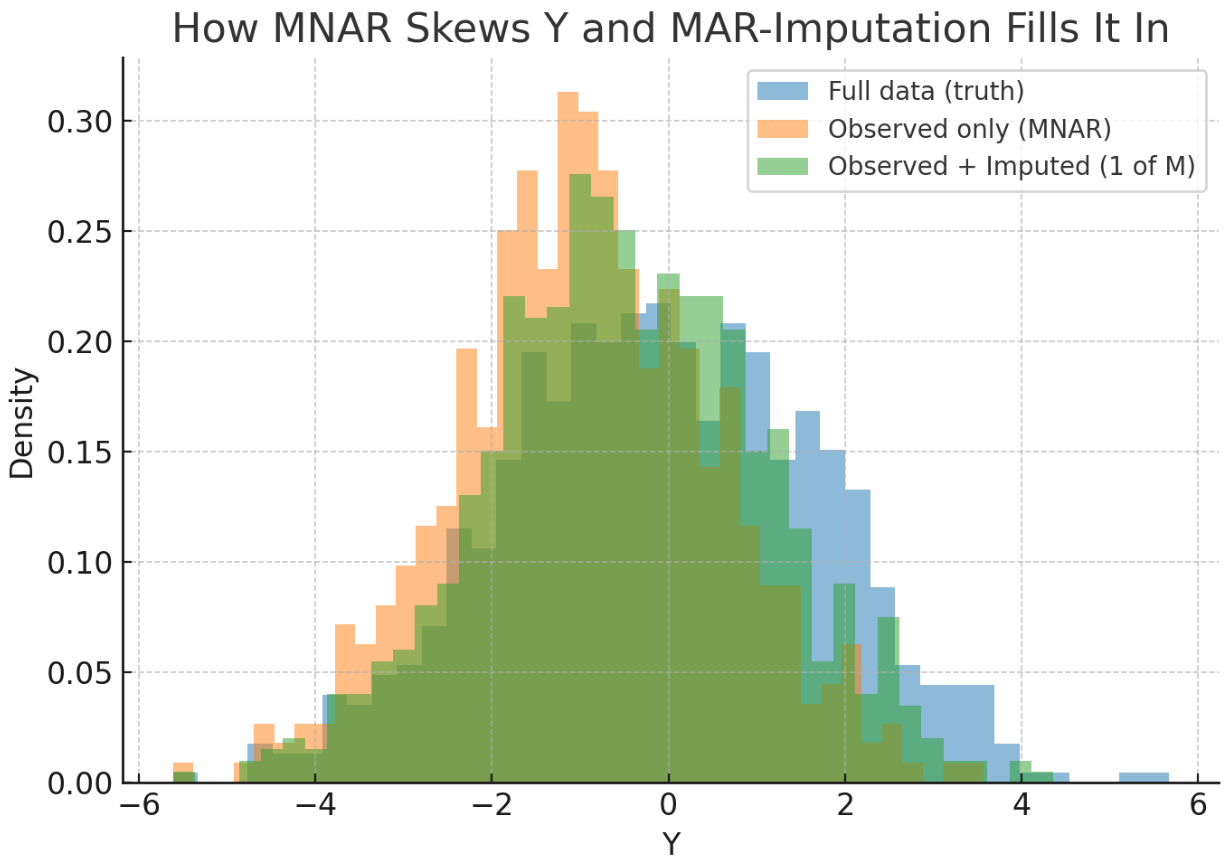
Q: Sure, first produce a simple simulation that visualizes MAR bias under MNAR.



MNAR Vs MAR-Imputation (Single Simulation)

	Quantity	Truth (full data)	Complete-case	MAR MI (pooled)	MNAR bias of MAR MI
1	Mean(Y)	-0.0838	-0.9417	-0.449	-0.3653
2	Slope (beta1)	1.4794	1.2593	1.2683	-0.2111
3	Intercept	-0.0412	-0.4186	-0.4125	-0.3713





Here’s a minimal simulation showing how **MNAR** missingness in YY induces **bias** when we impute under a (mis-specified) **MAR** model.

What I did (code + plots rendered above):

- Generated $X \sim \mathcal{N}(0,1)$ and $Y = \beta_0 + \beta_1 X + \varepsilon$ with $\beta_1 = 1.5$, $\varepsilon \sim \mathcal{N}(0,1)$.
- Induced **MNAR** missingness in Y: $P(R = 1 \mid Y) = \text{logit}^{-1}(a + gY)$ with g<0 so large Y values are **less likely observed**.
- Fit an OLS model on **observed** cases, then did **multiple imputation under MAR** (normal linear model draws for (β, σ^2)); pooled estimates across M=20 imputations (Rubin’s $\bar{q}$).

What you see:

- **Scatter/lines plot:**
  The **true line** (full data) vs the **complete-case (CCA)** fit vs the **MAR-MI pooled** fit. Both CCA and MAR-MI slopes fall **below** the truth because high-Y points go missing (MNAR), and MAR-based imputations can't fully reconstruct them.

- **Histograms:**
  The **observed** Y distribution is shifted left (MNAR skews what we see). The **observed+imputed** distribution (1 MI draw) partially fills the right tail but still **undershoots** the true right tail, so the mean stays biased low.

- **Table (displayed):**
  In this run (about 40% missing), the MAR-MI pooled estimates have noticeable **MNAR bias**:

  - MeanY: about -0.37 below truth

  - Slope: about -0.21 below truth

  - Intercept: about -0.37 below truth

Why the bias? Under MNAR, missingness depends on Y itself, so the correct conditional for $Y\_\text{mis} \mid X$ differs from the MAR model used to impute. Even with MI's between-imputation variability, the imputations are centered on the **wrong** regression surface and **don't rebuild** the suppressed right tail.

Want me to:

1. repeat this over many replications and summarize bias/coverage vs MNAR strength and missingness rate, or

2. add **delta-adjusted** (pattern-mixture) MI to show how sensitivity analysis recovers the right direction?

```
# Simulation to visualize MAR-based imputation bias when data are actually MNAR
# (Single run with MI under a mis-specified MAR assumption)
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from math import sqrt
from caas_jupyter_tools import display_dataframe_to_user

rng = np.random.default_rng(42)

# 1) Data-generating process
n = 800
beta0 = 0.0
beta1 = 1.5
sigma = 1.0

X = rng.normal(0, 1, size=n)
eps = rng.normal(0, sigma, size=n)
Y_full = beta0 + beta1 * X + eps

# Truth from full data
true_mean = float(Y_full.mean())
# population slope (approx via OLS on full data)
X1 = np.column_stack([np.ones(n), X])
beta_full = np.linalg.lstsq(X1, Y_full, rcond=None)[0]
```

```
true_intercept, true_slope = float(beta_full[0]), float(beta_full[1])

# 2) MNAR missingness in Y: P(R=1 | Y) = logistic(a + g*Y) with g<0 (high Y -> more missing)
# choose a to get ~40% missing
g = -1.0
# Find 'a' by simple search to hit target observed rate ~0.6
target_obs = 0.6
a_vals = np.linspace(-1.5, 1.5, 301)
obs_rates = []
for a in a_vals:
    p = 1 / (1 + np.exp(-(a + g*Y_full)))
    obs_rates.append(p.mean())
a = a_vals[np.argmin(np.abs(np.array(obs_rates) - target_obs))]

p_obs = 1 / (1 + np.exp(-(a + g*Y_full)))
R = rng.binomial(1, p_obs, size=n).astype(bool)

# Observed subset
X_obs, Y_obs = X[R], Y_full[R]
X_mis, Y_mis_true = X[~R], Y_full[~R]  # keep true Y for later comparison

# 3) Complete-case estimates
X1_obs = np.column_stack([np.ones(X_obs.shape[0]), X_obs])
```

```
beta_cc = np.linalg.lstsq(X1_obs, Y_obs, rcond=None)[0]
cc_intercept, cc_slope = float(beta_cc[0]), float(beta_cc[1])
cc_mean = float(Y_obs.mean())

# 4) MAR-based Multiple Imputation for missing Y | X using linear model fit on observed
#    We'll use a standard Normal-inverse-chi-square posterior draw for (beta, sigma^2)
M = 20

# OLS on observed data
n_obs = X_obs.shape[0]
p = 2
XtX = X1_obs.T @ X1_obs
XtX_inv = np.linalg.inv(XtX)
beta_hat = XtX_inv @ (X1_obs.T @ Y_obs)
resid = Y_obs - X1_obs @ beta_hat
s2_hat = (resid @ resid) / (n_obs - p)

# helper: random draws
def draw_beta_sigma():
    # sigma^2 ~ s2_hat * (n_obs - p) / chi2_{n_obs - p}
    chi2 = rng.chisquare(n_obs - p)
    sigma2 = s2_hat * (n_obs - p) / chi2
    # beta | sigma2 ~ Normal(beta_hat, sigma2 * (XtX)^(-1))
    L = np.linalg.cholesky(XtX_inv * sigma2)
    z = rng.normal(size=p)
    beta_draw = beta_hat + L @ z
    return beta_draw, sigma2

# Store MI estimates
slope_m = []
mean_m  = []
intercept_m = []

# Imputed datasets and a sample of imputed values for plotting
imputed_points = []

for m in range(M):
    beta_draw, sigma2_draw = draw_beta_sigma()
    # Impute Y_mis from Normal(mean = [1, X_mis] @ beta_draw, var = sigma2_draw)
    X1_mis = np.column_stack([np.ones(X_mis.shape[0]), X_mis])
    mu_mis = X1_mis @ beta_draw
    Y_mis_imp = rng.normal(mu_mis, sqrt(sigma2_draw))

    # Completed data
    Y_imp_all = Y_full.copy()
    Y_imp_all[~R] = Y_mis_imp

    # Estimates on completed data
    X1_all = np.column_stack([np.ones(n), X])
    b_imp = np.linalg.lstsq(X1_all, Y_imp_all, rcond=None)[0]
    intercept_m.append(float(b_imp[0]))
    slope_m.append(float(b_imp[1]))
    mean_m.append(float(Y_imp_all.mean()))

    # Save a subsample of imputed points for plotting
    if m == 0:
        imputed_points = pd.DataFrame({"X": X_mis, "Y_imp": Y_mis_imp})

# Rubin's rules
qbar_slope = float(np.mean(slope_m))
W_slope = float(np.var(slope_m, ddof=1) * 0)  # placeholder for within variance (not computed here per-imp)
B_slope = float(np.var(slope_m, ddof=1))

qbar_mean = float(np.mean(mean_m))
B_mean = float(np.var(mean_m, ddof=1))

qbar_intercept = float(np.mean(intercept_m))
B_intercept = float(np.var(intercept_m, ddof=1))
```

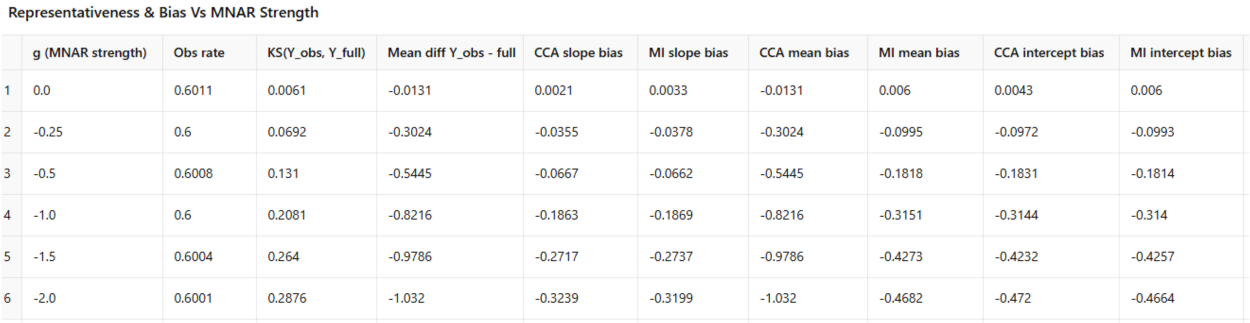
```
# Collect a summary table
summary = pd.DataFrame({
    "Quantity": ["Mean(Y)", "Slope (beta1)", "Intercept"],
    "Truth (full data)": [true_mean, true_slope, true_intercept],
    "Complete-case": [cc_mean, cc_slope, cc_intercept],
    "MAR MI (pooled)": [qbar_mean, qbar_slope, qbar_intercept],
    "MNAR bias of MAR MI": [qbar_mean - true_mean, qbar_slope - true_slope, qbar_intercept - true_intercept]
})
display_dataframe_to_user("MNAR vs MAR-Imputation (Single Simulation)", summary.round(4))

# 5) Plot 1: Observed vs Imputed scatter with true and fitted lines
plt.figure(figsize=(7,5))
plt.scatter(X_obs, Y_obs, alpha=0.5, label="Observed (R=1)")
plt.scatter(imputed_points["X"], imputed_points["Y_imp"], alpha=0.5, label="Imputed (1 of M)")
# True line
xs = np.linspace(X.min(), X.max(), 200)
plt.plot(xs, true_intercept + true_slope*xs, linewidth=2, label="True line (full data)")
# Complete-case fitted line
plt.plot(xs, cc_intercept + cc_slope*xs, linewidth=2, label="CCA fitted line")
# MAR MI pooled line
```

```
plt.plot(xs, qbar_intercept + qbar_slope*xs, linewidth=2, label="MAR MI pooled line")
plt.title("MNAR Missingness in Y; MAR-based Imputation")
plt.xlabel("X")
plt.ylabel("Y")
plt.legend()
plt.tight_layout()
plt.show()

# 6) Plot 2: Distribution of Y (full vs observed vs imputed-completed)
plt.figure(figsize=(7,5))
bins = 40
plt.hist(Y_full, bins=bins, alpha=0.5, density=True, label="Full data (truth)")
plt.hist(Y_obs, bins=bins, alpha=0.5, density=True, label="Observed only (MNAR)")
plt.hist(pd.concat([pd.Series(Y_obs), imputed_points["Y_imp"]]), bins=bins, alpha=0.5, density=True, label="Observed + Imputed (1 of M)")
plt.title("How MNAR Skews Y and MAR-Imputation Fills It In")
plt.xlabel("Y")
plt.ylabel("Density")
plt.legend()
plt.tight_layout()
plt.show()

# Provide a compact textual summary for the notebook output
```

summary

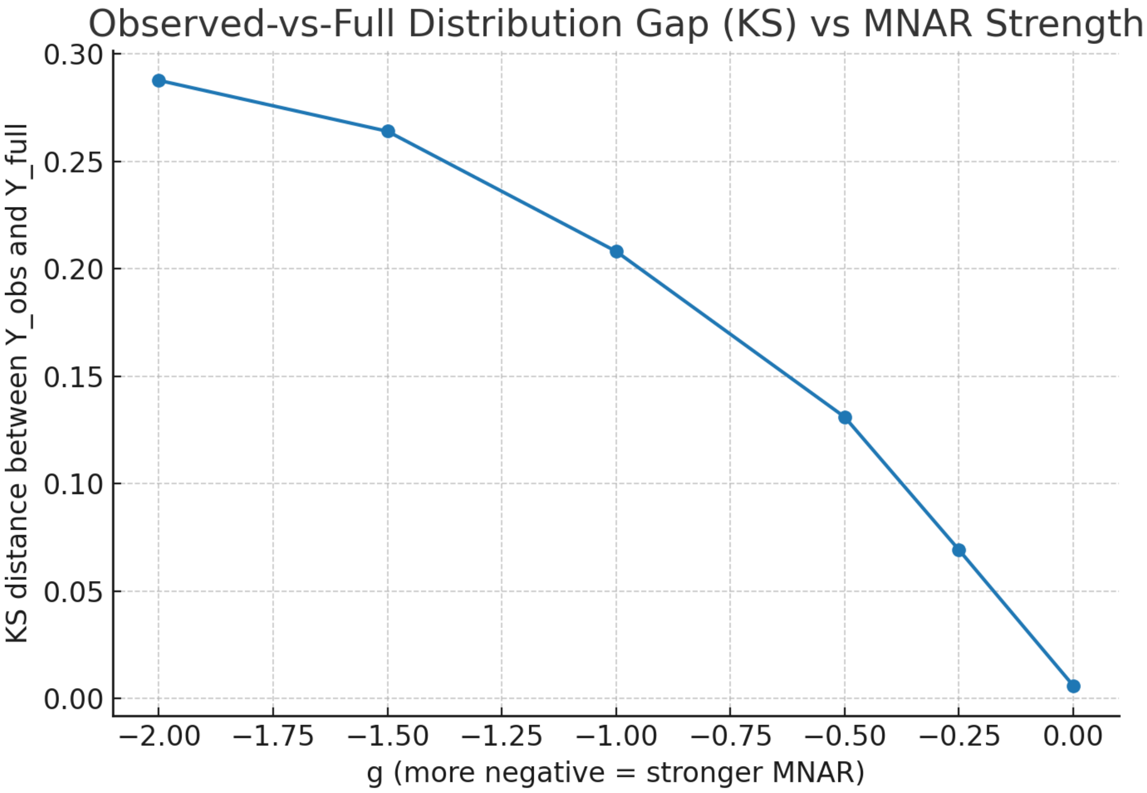
Comments: The AI takes some time but ultimately generates a simulation that isn’t overly burdensome or full of errors but instead functions correctly and visualizes the bias nicely. The regression slope plot in particular is fairly complicated but it is still done well by the AI. All of the plots are interpreted correctly (the AI appears to actually be “reading” them) along with the table results. All in all I think this response is well-done.

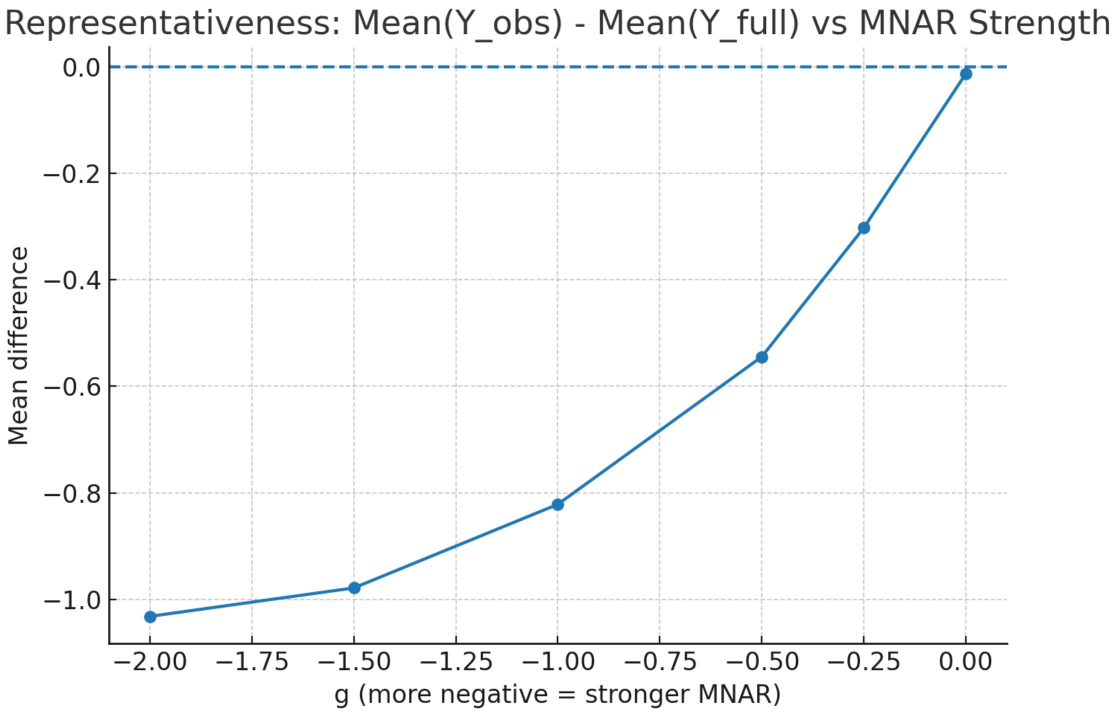
Q: Show how various levels of MNAR changes how well the observed cases "represent" the full data and compare biases.

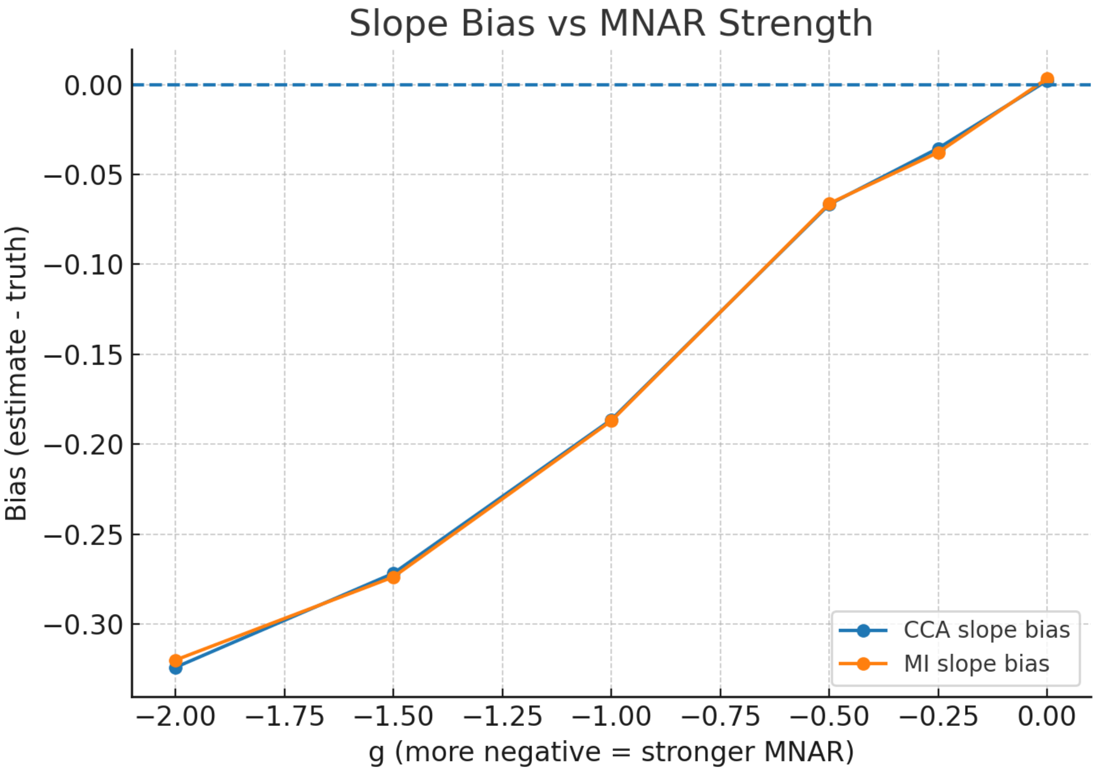


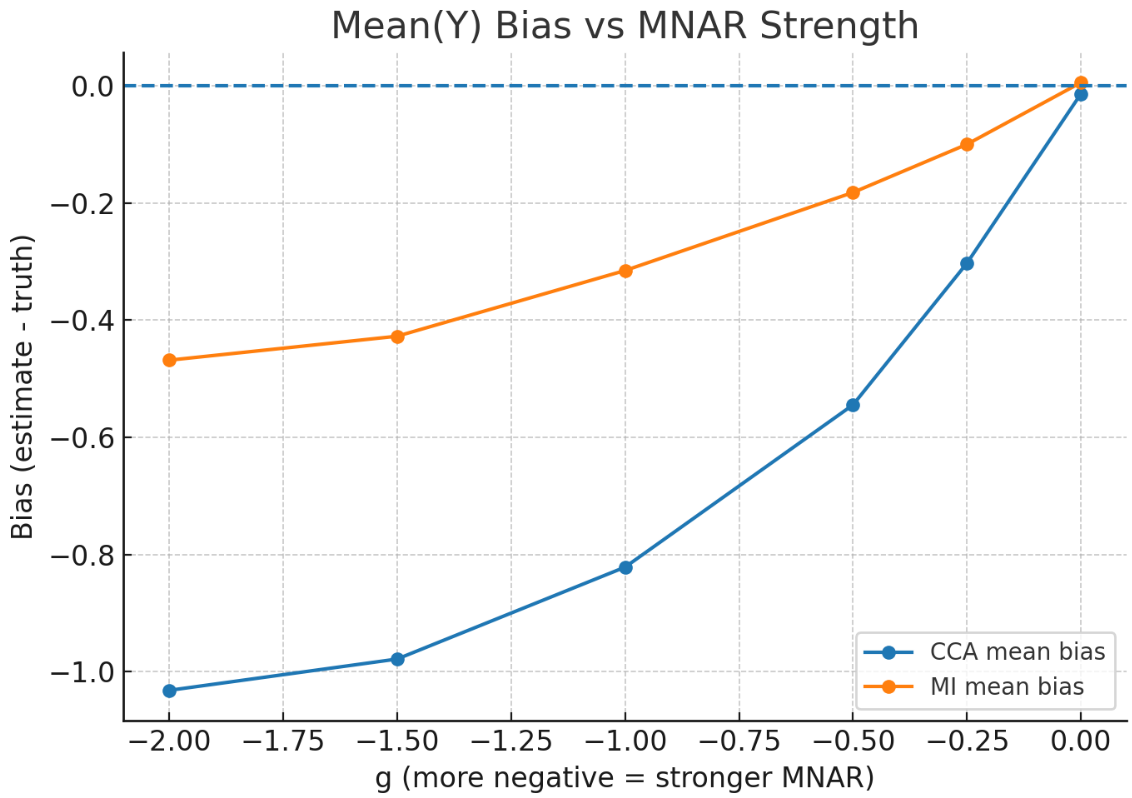
Representativeness & Bias Vs MNAR Strength

	g (MNAR strength)	Obs rate	KS(Y_obs, Y_full)	Mean diff Y_obs - full	CCA slope bias	MI slope bias	CCA mean bias	MI mean bias	CCA intercept bias	MI intercept bias
1	0.0	0.6011	0.0061	-0.0131	0.0021	0.0033	-0.0131	0.006	0.0043	0.006
2	-0.25	0.6	0.0692	-0.3024	-0.0355	-0.0378	-0.3024	-0.0995	-0.0972	-0.0993
3	-0.5	0.6008	0.131	-0.5445	-0.0667	-0.0662	-0.5445	-0.1818	-0.1831	-0.1814
4	-1.0	0.6	0.2081	-0.8216	-0.1863	-0.1869	-0.8216	-0.3151	-0.3144	-0.314
5	-1.5	0.6004	0.264	-0.9786	-0.2717	-0.2737	-0.9786	-0.4273	-0.4232	-0.4257
6	-2.0	0.6001	0.2876	-1.032	-0.3239	-0.3199	-1.032	-0.4682	-0.472	-0.4664




Representativeness: Mean(Y_obs) - Mean(Y_full) vs MNAR Strength
Mean difference
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
−2.00
−1.75
−1.50
−1.25
−1.00
−0.75
−0.50
−0.25
0.00
g (more negative = stronger MNAR)


Slope Bias vs MNAR Strength
Bias (estimate - truth)
0.00
−0.05
−0.10
−0.15
−0.20
−0.25
−0.30
−2.00
−1.75
−1.50
−1.25
−1.00
−0.75
−0.50
−0.25
0.00
g (more negative = stronger MNAR)
CCA slope bias
MI slope bias



I ran a sweep over MNAR strength g (with $P(R=1 \mid Y)=\operatorname{logit}^{-1}(a+gY)$). For **each** gg, I calibrated aa so the observed fraction stays ≈ 0.60; this isolates how *selection-on-Y* (not missingness rate) changes representativeness and bias.

What to look at in the outputs:

- The **table** (“Representativeness & Bias vs MNAR Strength”) shows, for each gg:
    1. **Representativeness** metrics: KS distance between Y_{obs} and Y_{full}, and mean difference $E[Y_{\text{obs}}]-E[Y]$.
    2. **Bias** for complete cases (CCA) and for **MAR-based MI** (normal linear model): biases of the slope and mean (also intercepts).
- The **plots** summarize the key patterns:

1. **KS distance ↑ with |g|**: as MNAR gets stronger (more negative g), the **observed distribution drifts** from the full distribution—observed cases become less representative.

2. **Observed mean shifts left**: $E[Y_{\text{obs}}] - E[Y]$ becomes more negative as gg decreases, because high-Y values are under-observed.

3. **Slope bias**: both **CCA** and **MAR-MI** slopes bias downward as MNAR strengthens (missing high-Y points flattens the line). MI helps only slightly; it cannot "see" the missing-right-tail under MNAR.

4. **Mean bias**: **MAR-MI** reduces the magnitude of mean bias relative to CCA, but remains biased downward; stronger MNAR → larger residual bias because MAR imputations are centered on the wrong conditional.

Takeaways:

- Holding the missingness rate fixed, **representativeness degrades monotonically with MNAR strength** (KS distance, mean shift).

- **MAR-based MI is not robust** to MNAR: it slightly mitigates, but does not remove, bias in both regression slope and mean.

- The **gap between CCA and MI** is largest for the mean (MI partially fills the suppressed tail); for the **slope**, both remain similarly biased as MNAR grows.

If you'd like, I can extend this to:

- vary the **missingness rate** (e.g., 20%, 40%, 60%) for each g,

- add **coverage** comparisons,

- or include a **delta-adjusted pattern-mixture MI** to show how sensitivity analysis can recover the right direction.

```
# MNAR strength sweep: representativeness of observed data and bias of CCA vs MAR-based MI
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from math import sqrt
from caas_jupyter_tools import display_dataframe_to_user

rng = np.random.default_rng(123)

# ----- helpers -----
def ks_statistic(a, b):
    """Two-sample Kolmogorov-Smirnov statistic (no SciPy)."""
    a = np.sort(np.asarray(a))
    b = np.sort(np.asarray(b))
    n = a.size
    m = b.size
    ia = 0
    ib = 0
    cdf_a = 0.0
    cdf_b = 0.0
    d = 0.0
    # merge-walk
    while ia < n and ib < m:
        if a[ia] <= b[ib]:
            x = a[ia]
            while ia < n and a[ia] == x:
                ia += 1
            cdf_a = ia / n
        else:
            x = b[ib]
            while ib < m and b[ib] == x:
                ib += 1
            cdf_b = ib / m
        d = max(d, abs(cdf_a - cdf_b))
    # flush tails
    if ia < n:
        d = max(d, abs(1.0 - cdf_b))
    if ib < m:
        d = max(d, abs(cdf_a - 1.0))
    return d


def calibrate_intercept_for_target(Y, g, target_obs=0.6, a_min=-4.0,
a_max=4.0, steps=801):
    """Choose 'a' so that mean P(R=1|Y) ≈ target_obs for a given g
and fixed Y."""
```

```
    a_vals = np.linspace(a_min, a_max, steps)
    ps = 1 / (1 + np.exp(-(a_vals[:,None] + g*Y[None,:])))
    obs_rates = ps.mean(axis=1)
    idx = np.argmin(np.abs(obs_rates - target_obs))
    return float(a_vals[idx]), float(obs_rates[idx])


def mi_under_mar(X, Y_full, R, M=20, rng=None):
    """Multiple imputation (linear-normal MAR model) for missing Y|X using observed cases."""
    if rng is None:
        rng = np.random.default_rng()
    X_obs, Y_obs = X[R], Y_full[R]
    n = X.size
    X1_all = np.column_stack([np.ones(n), X])
    X1_obs = np.column_stack([np.ones(X_obs.size), X_obs])
    # OLS on observed
    p = 2
    XtX = X1_obs.T @ X1_obs
    XtX_inv = np.linalg.inv(XtX)
    beta_hat = XtX_inv @ (X1_obs.T @ Y_obs)
    resid = Y_obs - X1_obs @ beta_hat
    s2_hat = (resid @ resid) / (Y_obs.size - p)

    def draw_beta_sigma():
```

```
    chi2 = rng.chisquare(Y_obs.size - p)
    sigma2 = s2_hat * (Y_obs.size - p) / chi2
    L = np.linalg.cholesky(XtX_inv * sigma2)
    z = rng.normal(size=p)
    beta_draw = beta_hat + L @ z
    return beta_draw, sigma2

slopes = []
means = []
intercepts = []

X_mis = X[~R]
X1_mis = np.column_stack([np.ones(X_mis.size), X_mis])

for _ in range(M):
    b_draw, s2_draw = draw_beta_sigma()
    mu_mis = X1_mis @ b_draw
    Y_imp = rng.normal(mu_mis, sqrt(s2_draw))
    Y_completed = Y_full.copy()
    Y_completed[~R] = Y_imp
    b_all = np.linalg.lstsq(X1_all, Y_completed, rcond=None)[0]
    intercepts.append(float(b_all[0]))
    slopes.append(float(b_all[1]))
    means.append(float(Y_completed.mean()))
    return np.mean(intercepts), np.mean(slopes), np.mean(means)

# ----- data generating process -----
n = 8000  # large n to reduce MC noise so trends are clear
beta0 = 0.0
beta1 = 1.5
sigma = 1.0
X = rng.normal(0, 1, size=n)
eps = rng.normal(0, sigma, size=n)
Y_full = beta0 + beta1 * X + eps

# "Truth" from full data
X1 = np.column_stack([np.ones(n), X])
b_full = np.linalg.lstsq(X1, Y_full, rcond=None)[0]
true_intercept, true_slope = float(b_full[0]), float(b_full[1])
true_mean = float(Y_full.mean())

# MNAR strengths (g): 0 is MCAR/MAR; negative increases MNAR severity
g_vals = [0.0, -0.25, -0.5, -1.0, -1.5, -2.0]

rows = []
target_obs = 0.6

for g in g_vals:
    a, achieved_obs = calibrate_intercept_for_target(Y_full, g=g, target_obs=target_obs)
    p_obs = 1 / (1 + np.exp(-(a + g*Y_full)))
    R = rng.binomial(1, p_obs, size=n).astype(bool)

    # Observed-only stats (representativeness)
    Y_obs = Y_full[R]
    ksY = ks_statistic(Y_obs, Y_full)
    mean_diff = float(Y_obs.mean() - true_mean)  # observed minus truth
    # CCA bias
    X1_obs = np.column_stack([np.ones(Y_obs.size), X[R]])
    b_cc = np.linalg.lstsq(X1_obs, Y_obs, rcond=None)[0]
    cc_intercept, cc_slope = float(b_cc[0]), float(b_cc[1])
    cc_mean = float(Y_obs.mean())
    cc_slope_bias = cc_slope - true_slope
    cc_mean_bias = cc_mean - true_mean
    cc_intercept_bias = cc_intercept - true_intercept

    # MI under MAR
    mi_intercept, mi_slope, mi_mean = mi_under_mar(X, Y_full, R, M=20, rng=rng)
    mi_slope_bias = mi_slope - true_slope
    mi_mean_bias = mi_mean - true_mean
    mi_intercept_bias = mi_intercept - true_intercept
    rows.append({
        "g (MNAR strength)": g,
        "Obs rate": achieved_obs,
        "KS(Y_obs, Y_full)": ksY,
        "Mean diff Y_obs - full": mean_diff,
        "CCA slope bias": cc_slope_bias,
        "MI slope bias": mi_slope_bias,
        "CCA mean bias": cc_mean_bias,
        "MI mean bias": mi_mean_bias,
        "CCA intercept bias": cc_intercept_bias,
        "MI intercept bias": mi_intercept_bias,
    })

summary = pd.DataFrame(rows)

# Show table
display_dataframe_to_user("Representativeness & Bias vs MNAR Strength", summary.round(4))

# ----- plots -----
# 1) KS distance vs MNAR strength
plt.figure(figsize=(7,5))
plt.plot(summary["g (MNAR strength)"], summary["KS(Y_obs, Y_full)"], marker="o")
plt.title("Observed-vs-Full Distribution Gap (KS) vs MNAR Strength")
plt.xlabel("g (more negative = stronger MNAR)")
plt.ylabel("KS distance between Y_obs and Y_full")
plt.tight_layout()
plt.show()


# 2) Mean difference (observed - full) vs MNAR strength
plt.figure(figsize=(7,5))
plt.plot(summary["g (MNAR strength)"], summary["Mean diff Y_obs - full"], marker="o")
plt.title("Representativeness: Mean(Y_obs) - Mean(Y_full) vs MNAR Strength")
plt.xlabel("g (more negative = stronger MNAR)")
plt.ylabel("Mean difference")
plt.axhline(0, linestyle="--")
plt.tight_layout()
plt.show()


# 3) Slope bias vs MNAR strength (CCA vs MI)
plt.figure(figsize=(7,5))
plt.plot(summary["g (MNAR strength)"], summary["CCA slope bias"], marker="o", label="CCA slope bias")
plt.plot(summary["g (MNAR strength)"], summary["MI slope bias"], marker="o", label="MI slope bias")
plt.title("Slope Bias vs MNAR Strength")
plt.xlabel("g (more negative = stronger MNAR)")
plt.ylabel("Bias (estimate - truth)")
plt.axhline(0, linestyle="--")
plt.legend()
plt.tight_layout()
plt.show()

# 4) Mean bias vs MNAR strength (CCA vs MI)
plt.figure(figsize=(7,5))
plt.plot(summary["g (MNAR strength)"], summary["CCA mean bias"], marker="o", label="CCA mean bias")
plt.plot(summary["g (MNAR strength)"], summary["MI mean bias"], marker="o", label="MI mean bias")
plt.title("Mean(Y) Bias vs MNAR Strength")
plt.xlabel("g (more negative = stronger MNAR)")
plt.ylabel("Bias (estimate - truth)")
plt.axhline(0, linestyle="--")
plt.legend()
plt.tight_layout()
plt.show()

"Done"
```