Handling Dependent Censoring in Survival Models: A Comparative Simulation Study

Efua Ainooson Noonoo

March 2025

Introduction

Traditional methods for analyzing censored survival data typically assume that censoring occurs independently of event times, implying the likelihood of an observation being censored is unrelated to the underlying survival distribution. However, in many practical situations, censoring is often dependent, aligning closely with data that are Missing Not At Random (MNAR). Under MNAR conditions, the probability of censoring systematically depends on unobserved or partially observed event times, introducing bias and potentially misleading estimates of survival probabilities and hazard ratios if not properly addressed Collett (2023); Little and Rubin (2019).

Dependent censoring, conceptualized as MNAR, frequently arises in clinical trials, epidemiological studies, and economic research where the censoring mechanism is inherently connected to the events under investigation. To correct biases arising from MNAR censoring, researchers commonly employ methods such as Inverse Probability of Censoring Weighting (IPCW). IPCW reweights uncensored observations to represent censored individuals with similar observed characteristics, thereby restoring missing information. By explicitly modeling the censoring distribution and assigning higher weights to subjects at greater risk of censoring, IPCW effectively mitigates bias induced by MNAR censoring processes Willems et al. (2023); Robins and Finkelstein (2000).

This project aims to do a comprehensive simulation study, comparing IPCW against alternative approaches, including naive analyses, selection models, and pattern mixture models Little (1993); Little and Rubin (2019), across multiple MNAR censoring conditions. Results from this comparative analysis will elucidate conditions under which IPCW is most effective, offering practical guidance for handling dependent censoring in survival analysis contexts

Methodology

A simulation study was conducted to investigate the effects of dependent censoring on survival analysis. Survival times were simulated from a log-normal distribution parameterized by covariates reflecting common clinical scenarios. Specifically, the covariates generated included age from a normal distribution with mean 60 and standard deviation 10, cholesterol from a lognormal distribution with mean log of 5.1 and standard deviation log of 0.3, hypertension and smoking status from Bernoulli distributions with probabilities 0.35 and 0.3 respectively, and exercise levels from an exponential distribution with a rate of 0.5. Survival times T_i were generated using:

$$T_i = \exp(lp_i + 0.5Z_i), \quad Z_i \sim N(0,1)$$

where the linear predictor lp_i was defined as:

$$lp_i = 3 - 0.03 \cdot age_i - 0.02 \cdot cholesterol_i - 0.5 \cdot hypertension_i - 0.7 \cdot smoking_i + 0.2 \cdot exercise_i.$$

Censoring times were generated under a Missing Not At Random (MNAR) mechanism, modeled as:

$$C_i = E_i \exp(-\phi T_i^2), \quad E_i \sim \text{Exp}(1),$$

where ϕ varied within $\{0, 0.2, 0.4, 0.6, 0.8, 1\}$, representing the degree of dependency between censoring and survival times. The observed time for each individual was $Y_i = \min(T_i, C_i)$, and the censoring indicator was defined as $\delta_i = I(T_i \leq C_i)$. Each scenario was replicated 50 times for sample sizes $n = 200, 500, 1000$.

Assessment of dependent censoring involved logistic regression modeling of the censoring indicator against survival times and covariates to detect significant relationships indicative of dependent censoring. Additionally, Weibull survival models were fitted separately for survival and censoring times to generate risk and censoring scores, respectively. Correlations between these scores were computed to quantify dependence further. Graphical methods, including density plots of survival times by censoring status and scatter plots of true versus observed survival times, supplemented these formal analyses.

To address the dependent censoring, several survival modeling techniques were compared. Initially, a naive Weibull model, which assumes independent censoring, was fitted as a baseline. Subsequently, an Inverse Probability of Censoring Weighting (IPCW) method Robins et al. (1994) was applied, where censoring probabilities were estimated from a Weibull censoring model, and stabilized weights were computed using Kaplan–Meier estimates to reduce variance. These weights were truncated at the 90th percentile to further minimize extreme weighting effects. The weighted Weibull regression model incorporated these stabilized weights to correct for dependent censoring.

In parallel, a Heckman selection model Heckman (1979) was implemented, employing a probit regression of the censoring indicator on covariates to compute the Inverse Mills Ratio (IMR), which was subsequently included as an additional covariate in a Weibull survival model. This approach explicitly adjusted for selection bias due to dependent censoring. Additionally, a parametric pattern-mixture model Little (1993) was applied, wherein data were stratified by censoring status, and separate Weibull survival models were fitted within each stratum. The parameter estimates from these strata were then combined using weighted averages based on the proportion of censored and observed cases.

Diagnostic evaluations of all models were conducted through standardized and deviance residual analyses, as well as goodness-of-fit assessments such as log–log survival plots. Correlation checks between risk scores and censoring scores were also performed to validate model adequacy. The performance of each modeling approach was evaluated using bias (the mean difference between estimated and true parameter values), mean squared error (MSE), and the Akaike Information Criterion (AIC), alongside comparisons of the estimated survival functions across different degrees of censoring dependence. Weibull survival models were specifically chosen due to their flexibility in parametric modeling of survival data Collett (2023).

Results

The simulation study was conducted with sample sizes n = 200, 500, 1000 across dependence parameters $\phi \in \{0, 0.2, 0.4, 0.6, 0.8, 1\}$, with 50 replications per scenario.

Since the simulation explicitly employed a Missing Not at Random (MNAR) censoring mechanism, censoring was expected to be informative regarding the survival times. Figure 1a reveals a clear difference in the distribution of true survival times between censored and uncensored individuals under the MNAR mechanism. Uncensored cases (in blue) exhibit a sharp peak at low survival times, while censored cases (in red) are distributed more broadly across longer survival durations. This contrast indicates that individuals with longer survival times are disproportionately likely to be censored. Such a pattern violates the assumption of independent censoring, as the censoring probability is clearly associated with the event time itself. This dependency introduces bias in standard survival analyses.

Figure 1c presents coefficient estimates for the naive and IPCW-weighted models across varying MNAR strengths (ϕ) and different sample sizes. For lower MNAR strengths ($\phi \leq 0.4$), both models yielded estimates close to the true parameter values. However, as the MNAR strength increased ($\phi > 0.5$), deviations from the true values became more apparent, particularly for the cholesterol covariate ($\beta_{\text{cholesterol}}$). Nevertheless, IPCW-weighted estimates remained consistently closer to the true parameter values compared to those from the naive model, emphasizing IPCW's effectiveness in adjusting for dependent censoring.

Figure 1d broadens this comparison to include selection and pattern-mixture models. The selection model consistently produced estimates farthest from the true parameter values, likely due to a violation of the normality assumption underlying its error terms. The pattern-mixture model yielded estimates that closely mirrored those from the selection model rather than the true parameter values. This discrepancy

may be due to the violation of the equal slopes assumption across censoring patterns, which necessitated stratification. As a result, the combined estimates may not have adequately recovered the true underlying effects. Conversely, IPCW-weighted and naive models generally resulted in estimates closer to the true values, with IPCW demonstrating superior performance overall.

To further evaluate model performance, the bias of each method was examined. Bias was defined as the difference between the estimated and true regression coefficients, averaged across replications. Figure 1b displays the average bias of the naive and IPCW-weighted models across varying MNAR strengths (ϕ) and sample sizes. The naive model consistently exhibited greater bias than the IPCW-weighted model, particularly as ϕ increased. This trend was most pronounced for the exercise and smoking covariates, where the naive model showed substantial upward or downward bias, while IPCW remained comparatively stable. Figure 1e extends the comparison to include the selection and pattern-mixture models. As anticipated, the IPCW-weighted model consistently reduced bias compared to the naive approach. The selection model demonstrated the highest bias among all methods. The pattern-mixture model produced biases similar to those of the selection model. These may be due to the violation.

To assess the precision of coefficient estimates, standard errors were calculated for each covariate across models and MNAR strengths. Figure 1f displays average standard errors by model and sample size for increasing values of ϕ. The IPCW-weighted Weibull model exhibited highly variable standard errors, with noticeable spikes occurring around $\phi \approx 0.6, 0.8$ and 0.4 for sample sizes 1000, 500 and 200 respectively for several covariates.

Truncating weights at the 90th percentile helped stabilize the other ϕ values for the IPCW standard errors, reducing extreme variability. In contrast, the naive, Heckman selection, and pattern-mixture models displayed consistently low and stable standard errors across all values of ϕ.

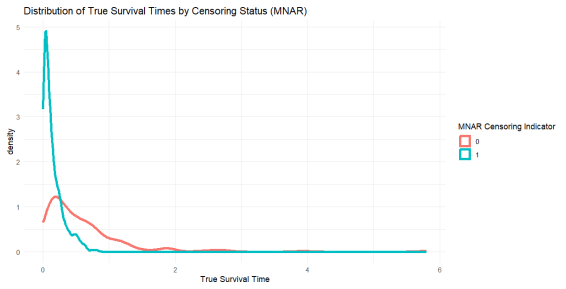
Despite truncation, the standard errors of the IPCW-weighted model remained higher than those of the other models. This reflects the additional variability introduced by estimating censoring weights, as discussed by Robins Robins et al. (1994). Specifically, the variance of the IPCW estimator can be expressed as:

$$\text{Var}\left(\hat{\beta}^{\text{IPCW}}\right) = \text{Var}\left(\hat{\beta}^{\text{naive}}\right) + \text{Var}_{\text{weights}}$$

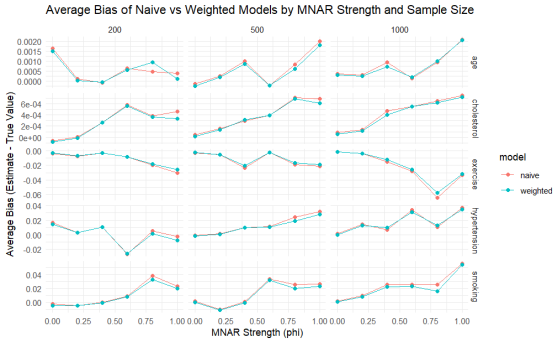
This increased variability results in wider confidence intervals for IPCW estimates, potentially reducing inferential precision. Nevertheless, while the naive, Heckman, and pattern-mixture models appear more precise, they fail to account for dependent censoring and thus remain biased. The IPCW-weighted model offers a bias-reducing alternative at the expense of higher standard errors.

To evaluate the overall accuracy of the coefficient estimates, Mean Squared Error (MSE) was computed as the sum of squared bias and variance. Figure 1g presents the MSE of the naive and IPCW-weighted models across MNAR strength (ϕ) and sample sizes. The naive model consistently exhibited higher MSE than the IPCW-weighted model across all covariates and settings, particularly at higher values of ϕ, where bias increased. These findings are reinforced in Figure 1h, which compares all four models. The IPCW-weighted model maintained the lowest MSE across covariates and MNAR conditions, confirming its advantage in handling dependent censoring. The selection model showed the highest MSE overall, likely due to its poor performance under assumption violations. While the pattern-mixture and naive models had intermediate MSE levels, they were still outperformed by IPCW. Notably, all models exhibited relatively low MSEs (below 0.25), with the IPCW-weighted model showing the best balance of bias and variance.

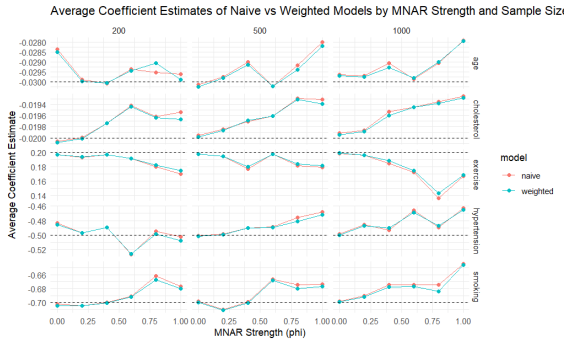
Finally, the Akaike Information Criterion (AIC) was used to compare model fit between the naive and IPCW-weighted Weibull models. Figure 2a displays the AIC values across MNAR strength (ϕ) and sample sizes. For $\phi < 0.75$, the IPCW-weighted model consistently produced lower AIC values than the naive model, suggesting improved fit under moderate levels of dependent censoring. As ϕ and sample size increased, the AIC values of both models became more variable, with the naive model showing notable instability—including extreme negative AIC values at higher ϕ. This may reflect poor likelihood behavior in the presence of strong dependence and limited information from censored cases. In contrast, the IPCW-weighted model maintained more stable AIC values, even under strong dependence scenarios, indicating its robustness when correcting for dependent censoring. However, it is important to note that the AIC returned for the IPCW-weighted model is based on a weighted pseudo-likelihood rather than a full likelihood. As such, the resulting AIC values are not strictly comparable in the classical sense and should be interpreted with caution. Despite this limitation, the comparison still provides a useful heuristic for evaluating relative model fit.



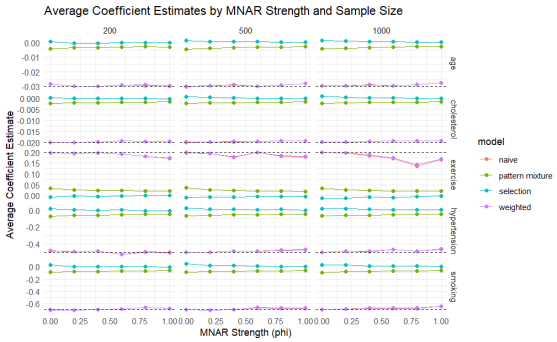
(a) Survival Times by Censoring Status



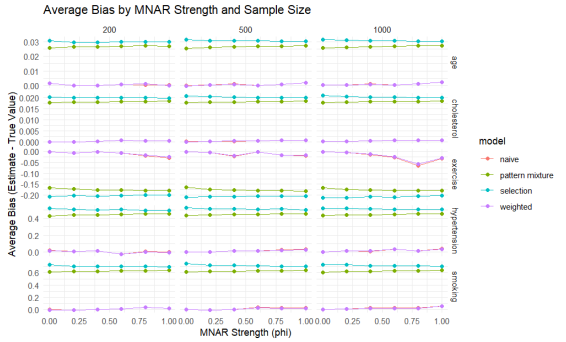
(b) Bias: naive vs. weighted



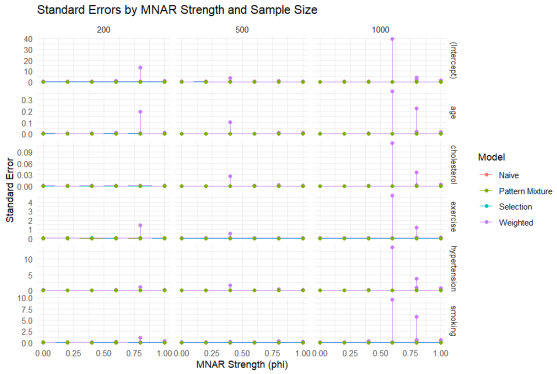
(c) Average Coefficients: naive vs. weighted



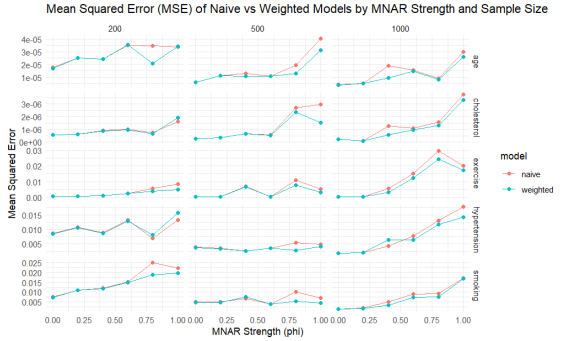
(d) Average Coefficient Estimates



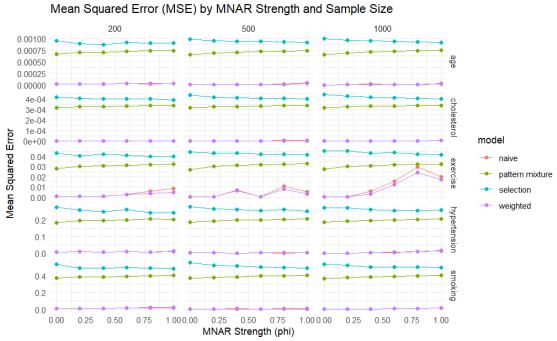
(e) Bias of three methods



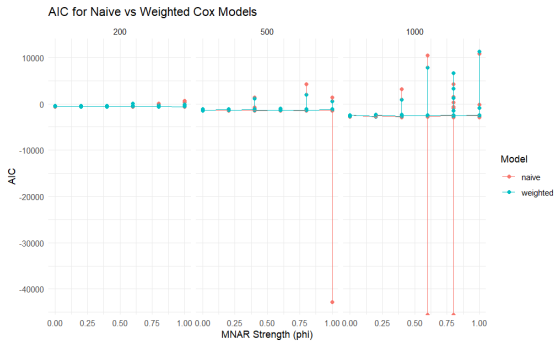
(f) Standard Errors across Models



(g) MSE: Naive vs. Weighted



(h) Overall MSE



(a) AIC for Naive vs. IPCW-Weighted Weibull Models by MNAR Strength and Sample Size.

Discussion

The findings provide valuable insights into the trade-offs between bias, variance, and model fit, offering practical guidance for survival analysis under dependent censoring. Overall, the IPCW-weighted Weibull model emerged as the most robust approach when the censoring mechanism was moderately dependent on survival time ($\phi < 0.75$). Its consistent ability to reduce bias and mean squared error (MSE) across scenarios makes it a more accurate method for estimating covariate effects an important consideration in clinical trials and epidemiological studies where dependent censoring is often encountered.

However, the increased variability introduced by IPCW weights, even after applying truncation at the 90th percentile, led to larger standard errors and wider confidence intervals. This variance inflation reflects the known trade-off between bias correction and estimation precision. The naive model, although more stable in terms of standard error, exhibited increased MSE and volatile AIC values at higher levels of ϕ, underscoring its limitations when the assumption of independent censoring is violated.

The Heckman selection model performed poorly in this context, likely due to the violation of the normality assumption in the censoring mechanism. Similarly, the pattern-mixture model showed limited stability, particularly under strong dependence, and required stratification due to non-parallel effects across censoring strata. These findings suggest that, while appealing in theory, these alternative models may be less reliable in practice when key assumptions are not satisfied.

Several limitations should be acknowledged. First, the simulation study was based on a specific data-generating mechanism, which may not capture the full complexity of real-world survival data. Future work should evaluate these models in applied settings with known or suspected dependent censoring. Second, while truncating IPCW weights at the 90th percentile improved model stability, optimal truncation thresholds may be dataset-specific and warrant further investigation. Third, the instability observed in the naive model's AIC at higher ϕ suggests that additional simulation replications could help clarify whether this reflects genuine model inadequacy or sampling variability.

Lastly, this study focused on coefficient estimation rather than predictive performance, which is often of primary interest in survival applications. Future research should incorporate prediction-based metrics to assess models more comprehensively.

Conclusion

In conclusion, the IPCW-weighted Weibull model provides a robust approach for addressing dependent censoring in survival analysis, particularly under moderate MNAR strength. Its ability to reduce bias and improve overall accuracy makes it a valuable tool in settings where the independence assumption is violated. However, the increase in variance reflected in wider standard errors highlight the importance of model choice under dependent censoring, especially in small samples. The Heckman selection model should be applied with caution due to its reliance on strong distributional assumptions, particularly the normality of error terms. While the pattern-mixture model did not perform as well as IPCW in this study, it may offer a viable alternative when the data support stratification and when its assumptions are met.

Appendix

References

Collett, D. (2023). *Modelling Survival Data in Medical Research*. 4th edition. Chapman and Hall/CRC.

Cole, S. R., & Hernán, M. A. (2008). Constructing inverse probability weights for marginal structural models. *American Journal of Epidemiology*, 168(6), 656–664.

Fleming, T. R., & Harrington, D. P. (1991). *Counting Processes and Survival Analysis*. Wiley.

Heckman, J. J. (1979). Sample Selection Bias as a Specification Error. *Econometrica*, 47(1), 153–161.

Klein, J. P., & Moeschberger, M. L. (2003). *Survival Analysis: Techniques for Censored and Truncated Data*. 2nd edition. Springer.

Little, R. J. A. (1993). Pattern-mixture models for multivariate incomplete data. *Journal of the American Statistical Association*, 88(421), 125–134.

Little, R. J. A., & Rubin, D. B. (2019). *Statistical Analysis with Missing Data*, 3rd edition. Wiley.

R Core Team (2023). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL: `https://www.R-project.org/`.

Robins, J. M., Rotnitzky, A., & Zhao, L. P. (1994). Estimation of regression coefficients when some regressors are not always observed. *Journal of the American Statistical Association*, 89(427), 846–866.

Robins, J. M., & Finkelstein, D. M. (2000). Correcting for noncompliance and dependent censoring in an AIDS clinical trial with inverse probability of censoring weighted (IPCW) log-rank tests. *Biometrics*, 56(3), 779–788.

Therneau, T. M., & Grambsch, P. M. (2000). *Modeling Survival Data: Extending the Cox Model*. Springer.

Willems, S. J. W., Schat, A., van Noorden, M. S., & Fiocco, M. (2023). Correcting for dependent censoring in routine outcome monitoring data by applying the inverse probability censoring weighted estimator. *Statistical Methods in Medical Research*, 32(3), 568–581.

R CODE

```r


library(survival)
library(ggplot2)
library(tidyr)
library(dplyr)


set.seed(4321)

simulate_mnar = function(phi, n ){



## 1. Covariate Simulation

age = rnorm(n, mean = 60, sd = 10)

cholesterol = rlnorm(n, meanlog = 5.1, sdlog = 0.3)

hypertension = rbinom(n, size = 1, prob = 0.35)

smoking = rbinom(n, size = 1, prob = 0.3)

exercise = rexp(n, rate = 0.5)

# 2) Generate Lognormal Survival Times (for mis-specification)

  Z = rnorm(n)
  lp = 3 - 0.03*age - 0.02*cholesterol - 0.5*hypertension - 0.7*smoking + 0.2*
      exercise
  survival_time = exp(lp + 0.5*Z)

 ## 3. Generate Censoring Times

censor_time_mar = rexp(n, rate = 1)* (1 + 0.02 * age)

censor_time_mnar = rexp(n, rate = 1) * exp(-phi * survival_time^2)


## 4. Apply Censoring: MAR and MNAR

## Different censoring scenarios

### MAR

censor_time_mar_applied = pmin(survival_time,censor_time_mar)

observed_time_mar = pmin(survival_time,censor_time_mar_applied)

censor_indicator_mar = as.numeric(survival_time <= censor_time_mar_applied)

### MNAR

censor_time_mnar_applied = pmin(survival_time,censor_time_mnar)

```

```r
observed_time_mnar = pmin(survival_time,censor_time_mnar_applied)

censor_indicator_mnar = as.numeric(survival_time <= censor_time_mnar_applied)


sim.data = data.frame(
  age = age,
  cholesterol = cholesterol,
  hypertension = hypertension,
  smoking = smoking,
  exercise = exercise,
  survival_time = survival_time,
  censor_time_mar = censor_time_mar,
  censor_time_mnar = censor_time_mnar,
  censor_time_mar_applied = censor_time_mar_applied,
  observed_time_mar = observed_time_mar,
  censor_indicator_mar = censor_indicator_mar,
  censor_time_mnar_applied = censor_time_mnar_applied,
  observed_time_mnar = observed_time_mnar,
  censor_indicator_mnar = censor_indicator_mnar
)

### Inspection
## Table
table1 = table(censor_indicator_mar) / n
table2 = table(censor_indicator_mnar) / n

 ## 6. Visual Inspection Plots

## Visual Inspection

visual.inspection_plot1 = ggplot(sim.data, aes(x = survival_time, colour = as.
    factor(censor_indicator_mnar)))+
  geom_density(size = 1.5)+
  labs(
    title = "Distribution of True Survival Times by Censoring Status (MNAR)",
    x = "True Survival Time",
    color = "MNAR Censoring Indicator"
  ) +
  theme_minimal()

visual.inspection_plot2 = ggplot(sim.data, aes(x = survival_time , y = observed_
    time_mnar, color = as.factor(censor_indicator_mnar)))+
  geom_point(alpha = 2) +
  labs(title = "Observed Time vs. True Survival Time (MNAR Scenario)",
       x = "True Survival Time (T)",
       y = "Observed Survival Time",
       color = "Censoring Status (MNAR)")+
  theme_minimal()



  ## 7. Further Analysis (Logistic regression for censoring indicator)


furthur.analysis_model = glm(censor_indicator_mnar~survival_time, data = sim.data,
     family = binomial)

summary(furthur.analysis_model)
```

```r


  ## A 8. Naive Weibull Model (ignoring dependent censoring)

naive_model = survreg(Surv(observed_time_mnar, censor_indicator_mnar) ~
                      age + cholesterol + hypertension + smoking + exercise, data
                          = sim.data)

summary(naive_model)

naive_sum = summary(naive_model)$coefficients
naive_ci = confint(naive_model)



## 9. Fit a Parametric Survival Model (for risk scores)

model2 = survreg(Surv(observed_time_mnar,censor_indicator_mnar)~ age + cholesterol
                 + hypertension + smoking + exercise, data =sim.data)

summary(model2)



## 10. Fit a Parametric Censoring Model (Weibull) using survreg
## Note: We use (1 - censor_indicator_mnar) so that event = 1 indicates being
    censored.
## Time to censoring fitted by a weibull distribution to obtain censoring score.


model3 = survreg(Surv(censor_time_mnar, 1 - censor_indicator_mnar) ~ age +
    cholesterol + hypertension + smoking + exercise,
                         data = sim.data)

summary(model3)


 ## 11. Plot Risk Score vs. Censoring Score

# risk score for each participant
risk_score = predict(model2, type = "lp")

# censoring score for each participant
censoring_score = predict(model3, type = "lp")

#plot(risk_score, censoring_score)

score_data = data.frame(risk_score = risk_score,
                        censoring_score=censoring_score)

risk.score_vs_censoring_score_plot = ggplot(data = score_data, aes(x = censoring_
    score, y = risk_score)) +
  geom_point(alpha = 0.8, color = "blue") +
  labs(title = "Risk Score vs. Censoring Score",
       x = "Censoring Score",
       y = "Risk Score",
       color = "Group") +
  theme_minimal()
```

```r


### Correlation value

correlation_value = cor.test(risk_score, censoring_score)


## 12. Weight Calculation using Cox for MNAR
# Fit a censoring model using Weibull

model4 = survreg(Surv(censor_time_mnar, 1 - censor_indicator_mnar) ~
                 age + cholesterol + hypertension + smoking + exercise,
                 data = sim.data)

# Calculate survival probabilities at observed times using predict with survreg
n_rows = nrow(sim.data)
S_ci = numeric(n_rows)
for (i in seq_len(n_rows)) {
  lp = predict(model4, newdata = sim.data[i, ], type = "lp")
  S_ci[i] = 1 - pweibull(sim.data$observed_time_mnar[i],
                         shape = 1/model4$scale,
                         scale = exp(lp))
}

# Basic weights (inverse probability weights)
weight = 1 / S_ci
sim.data$weight = weight
summary(sim.data$weight)

## 13. Stabilized Weight Calculation
# Fit a nonparametric KM estimate of the censoring survival function using the
    same data

model5 = survfit(Surv(censor_time_mnar, 1 - censor_indicator_mnar) ~ 1, data = sim
    .data)

n_rows = nrow(sim.data)
S_KM_t = numeric(n_rows)
for (i in seq_len(n_rows)) {
  surv_info = summary(model5, times = sim.data$observed_time_mnar[i], extend =
      TRUE)$surv
  S_KM_t[i] = surv_info[1]
}

epsilon = 1e-8
S_KM_t_adj = pmax(S_KM_t, epsilon)
S_ci_adj   = pmax(S_ci, epsilon)

weight_star = S_KM_t_adj / S_ci_adj
sim.data$weight_star = weight_star
summary(sim.data$weight_star)

# Compute truncated weights and assign
## truncation at 90 because of the weight of the distribution
tmp = pmin(sim.data$weight_star, quantile(sim.data$weight_star, 0.90, na.rm = TRUE
    ))
sim.data$weight_star_trunc = tmp
weight_star_trunc = tmp

```

```r

## B 14. Weighted Weibull Model using Stabilized Weights

weighted_model = survreg(Surv(observed_time_mnar, censor_indicator_mnar) ~
                      age + cholesterol + hypertension + smoking + exercise,
                      weights = weight_star_trunc,
                      data = sim.data,
                      robust = TRUE)

summary(weighted_model)

weighted_sum = summary(weighted_model)$coefficients
  weighted_ci = confint(weighted_model)


#### C. SELECTION MODEL

  ### 1. Probit Model for Censoring Indicator

probit_model = glm(censor_indicator_mnar~age + cholesterol + hypertension +
    smoking + exercise, family = binomial(link = "probit"), data = sim.data)

summary(probit_model)

### 2. Inverse Mills Ratio (IMR)

xgamma = predict(probit_model, type = "link")

phi_xgamma = dnorm(xgamma)

Phi_xgamma = pnorm(xgamma)

IMR = numeric(nrow(sim.data))

IMR[sim.data$censor_indicator_mnar == 1] = phi_xgamma[sim.data$censor_indicator_
    mnar == 1] /Phi_xgamma[sim.data$censor_indicator_mnar == 1]

## conditional expectation of z

cond.exp_z = xgamma + phi_xgamma/Phi_xgamma

### 3. Outcome Model (with IMR)

selected_data = sim.data[sim.data$censor_indicator_mnar == 1, ]
selected_data$IMR = IMR[sim.data$censor_indicator_mnar == 1]

heckman_model = lm(observed_time_mnar~age + cholesterol + hypertension + smoking +
     exercise + IMR,  data = selected_data)

summary(heckman_model)


#### D. Pattern-Mixture Models

### 1. Creating Subgroups

observed_data = sim.data[sim.data$censor_indicator_mnar == 1, ]
censored_data = sim.data[sim.data$censor_indicator_mnar == 0, ]

```

```r
pattern = factor(sim.data$censor_indicator_mnar, levels = c(0, 1),
                 labels = c("Censored", "Observed"))
sim.data$pattern = pattern

### 2. Seperate linear models for group

lm_observed = lm(observed_time_mnar ~ age + cholesterol + hypertension + smoking + exercise,
                 data = subset(sim.data, pattern == "Observed"))

lm_censored = lm(observed_time_mnar ~ age + cholesterol + hypertension + smoking + exercise,
                 data = subset(sim.data, pattern == "Censored"))

summary(lm_observed)$coefficients
summary(lm_censored)$coefficients

anova(reduced_model, full_model)

### 3. Pattern weights

p_observed = nrow(subset(sim.data, pattern == "Observed")) / nrow(sim.data)

p_censored = nrow(subset(sim.data, pattern == "Censored")) / nrow(sim.data)


### 4. Combined Coefficients

coef_combined = (p_observed * coef(lm_observed)) + (p_censored * coef(lm_censored))
coef_combined

### 5. Standard Errors using Delta Method

coef_obs = coef(lm_observed)
coef_cens = coef(lm_censored)
vcov_obs = vcov(lm_observed)
vcov_cens = vcov(lm_censored)

var_combined = p_observed^2 * diag(vcov_obs) + p_censored^2 * diag(vcov_cens)
se_combined = sqrt(var_combined)

###5. Confidence Intervals

z = qnorm(0.975)
lower_ci = coef_combined - z * se_combined
upper_ci = coef_combined + z * se_combined


## Summary table

pattern_mixture_summary = data.frame(
  Estimate = coef_combined,
  SE = se_combined,
  LowerCI = lower_ci,
  UpperCI = upper_ci

)

```

```r
 sim_results = list(phi = phi,
       naive = list(coefficients = naive_sum, ci = naive_ci),
       weighted = list(coefficients = weighted_sum, ci = weighted_ci),
      naive_model = naive_model,
      weighted_model = weighted_model,
      furthur.analysis_model = summary(furthur.analysis_model),
      visual.inspection_plot1 = visual.inspection_plot1,
      visual.inspection_plot2 = visual.inspection_plot2,
      correlation_value = correlation_value,
      risk.score_vs_censoring_score_plot =risk.score_vs_censoring_score_plot,
      table2 = table2,
      weight_star_trunc = weight_star_trunc,
      heckman_model = heckman_model,
      pattern_mixture = list(
      coefficients = coef_combined,
      standard_errors = se_combined,
      ci = pattern_mixture_summary[, c("LowerCI", "UpperCI")]
    ),
     sim.data = sim.data

      )

 return(sim_results)

}

## REPLICATION
set.seed(4321)
n = c(200,500,1000)
phi_values = seq(0,1,0.2)
n_reps = 50


results_list = list()
idx = 1



for (i in n) {
  for (phi in phi_values) {
    for (rep in 1:n_reps) {
      res = tryCatch(simulate_mnar(phi = phi, n = i),
                     error = function(e) {
                       message(sprintf("Error for phi = %s, sample size = %s,
                           replicate = %s: %s", phi, i, rep, e$message))
                       return(NULL)
                     })
      if (is.null(res)) next
      res$sample_size = i
      res$phi = phi
      res$replicate = rep
      results_list[[idx]] = res
      idx = idx + 1
    }
  }
}

## 8. Data Frame
results_df = do.call(rbind, lapply(results_list, function(res) {
```

```r

  common_cols = Reduce(intersect, list(
    colnames(as.data.frame(res$naive$coefficients)),
    colnames(as.data.frame(res$weighted$coefficients)),
    colnames(as.data.frame(res$heckman$coefficients)),
    colnames(as.data.frame(res$pattern_mixture$coefficients))
  ))

  # Naive model
  naive_df = as.data.frame(res$naive$coefficients)
  naive_df = naive_df[, common_cols, drop = FALSE]
  naive_df$variable = rownames(naive_df)
  naive_df$model = "naive"
  naive_df$phi = res$phi
  naive_df$sample_size = rep(res$sample_size, nrow(naive_df))

  naive_ci_df = as.data.frame(res$naive$ci)
  if (is.null(naive_ci_df) || ncol(naive_ci_df) == 0 || nrow(naive_ci_df) == 0) {
    naive_ci_df = data.frame(lower_ci = rep(NA, nrow(naive_df)),
                             upper_ci = rep(NA, nrow(naive_df)))
  } else {
    colnames(naive_ci_df) = c("lower_ci", "upper_ci")
  }
  naive_df$lower_ci = naive_ci_df$lower_ci
  naive_df$upper_ci = naive_ci_df$upper_ci

  # Weighted model
  weighted_df = as.data.frame(res$weighted$coefficients)
  weighted_df = weighted_df[, common_cols, drop = FALSE]
  weighted_df$variable = rownames(weighted_df)
  weighted_df$model = "weighted"
  weighted_df$phi = res$phi
  weighted_df$sample_size = rep(res$sample_size, nrow(weighted_df))

  weighted_ci_df = as.data.frame(res$weighted$ci)
  if (is.null(weighted_ci_df) || ncol(weighted_ci_df) == 0 || nrow(weighted_ci_df)
      == 0) {
    weighted_ci_df = data.frame(lower_ci = rep(NA, nrow(weighted_df)),
                                upper_ci = rep(NA, nrow(weighted_df)))
  } else {
    colnames(weighted_ci_df) = c("lower_ci", "upper_ci")
  }
  weighted_df$lower_ci = weighted_ci_df$lower_ci
  weighted_df$upper_ci = weighted_ci_df$upper_ci

  ## Heckman model

  heckman_df = as.data.frame(res$heckman$coefficients)
  heckman_df = heckman_df[, common_cols, drop = FALSE]
  heckman_df$variable = rownames(heckman_df)
  heckman_df$model = "heckman"
  heckman_df$phi = res$phi
  heckman_df$sample_size = rep(res$sample_size, nrow(heckman_df))

  heckman_ci_df = as.data.frame(res$heckman$ci)
  if (is.null(heckman_ci_df) || ncol(heckman_ci_df) == 0 || nrow(heckman_ci_df) ==
      0) {
    heckman_ci_df = data.frame(lower_ci = rep(NA, nrow(heckman_df)),
                               upper_ci = rep(NA, nrow(heckman_df)))
```

```
  } else {
    colnames(heckman_ci_df) = c("lower_ci", "upper_ci")
  }
  heckman_df$lower_ci = heckman_ci_df$lower_ci
  heckman_df$upper_ci = heckman_ci_df$upper_ci

  ## Pattern mixture model

  pattern_mixture_df = as.data.frame(res$pattern_mixture$coefficients)
  pattern_mixture_df = pattern_mixture_df[, common_cols, drop = FALSE]
  pattern_mixture_df$variable = rownames(pattern_mixture_df)
  pattern_mixture_df$model = "Pattern Mixture"
  pattern_mixture_df$phi = res$phi
  pattern_mixture_df$sample_size = rep(res$sample_size, nrow(pattern_mixture_df))

  pattern_mixture_ci_df = as.data.frame(res$pattern_mixture$ci)
  if (is.null(pattern_mixture_ci_df) || ncol(pattern_mixture_ci_df) == 0 || nrow(
      pattern_mixture_ci_df) == 0) {
    pattern_mixture_ci_df = data.frame(lower_ci = rep(NA, nrow(pattern_mixture_df)
      ),
                                       upper_ci = rep(NA, nrow(pattern_mixture_df)))
  } else {
    colnames(pattern_mixture_ci_df) = c("lower_ci", "upper_ci")
  }
  pattern_mixture_df$lower_ci = pattern_mixture_ci_df$lower_ci
  pattern_mixture_df$upper_ci = pattern_mixture_ci_df$upper_ci

  rbind(naive_df, weighted_df, heckman_df, pattern_mixture_df)
}))


#DIAGNOSTIC CHECK

print(results_list[[1]]$correlation_value)
print(results_list[[1]]$visual.inspection_plot1)
print(results_list[[1]]$visual.inspection_plot2)
print(results_list[[1]]$risk.score_vs_censoring_score_plot)


### PLOTS FOR COMPARISON

#  Data Frame for Coefficient Estimates
results_df = do.call(rbind, lapply(results_list, function(res) {

  # Na ve model summary
  naive_df = tryCatch({
    data.frame(
      variable    = names(res$naive$coefficients),
      Estimate    = as.numeric(res$naive$coefficients),
      lower_ci    = res$naive$ci[,1],
      upper_ci    = res$naive$ci[,2],
      model       = "naive",
      phi         = res$phi,
      sample_size = res$sample_size,
      replicate   = res$replicate,
      stringsAsFactors = FALSE
    )
  }, error = function(e) {
    data.frame(
```

```
      variable     = names(res$naive$coefficients),
      Estimate     = as.numeric(res$naive$coefficients),
      lower_ci     = rep(NA, length(res$naive$coefficients)),
      upper_ci     = rep(NA, length(res$naive$coefficients)),
      model        = "naive",
      phi          = res$phi,
      sample_size  = res$sample_size,
      replicate    = res$replicate,
      stringsAsFactors = FALSE
    )
  })

  # Weighted model summary
  weighted_df = tryCatch({
    data.frame(
      variable     = names(res$weighted$coefficients),
      Estimate     = as.numeric(res$weighted$coefficients),
      lower_ci     = res$weighted$ci[,1],
      upper_ci     = res$weighted$ci[,2],
      model        = "weighted",
      phi          = res$phi,
      sample_size  = res$sample_size,
      replicate    = res$replicate,
      stringsAsFactors = FALSE
    )
  }, error = function(e) {
    data.frame(
      variable     = names(res$weighted$coefficients),
      Estimate     = as.numeric(res$weighted$coefficients),
      lower_ci     = rep(NA, length(res$weighted$coefficients)),
      upper_ci     = rep(NA, length(res$weighted$coefficients)),
      model        = "weighted",
      phi          = res$phi,
      sample_size  = res$sample_size,
      replicate    = res$replicate,
      stringsAsFactors = FALSE
    )
  })

  # Heckman Selection model summary
  res$heckman$coefficients = res$heckman$coefficients[names(res$heckman$
      coefficients) != "IMR"]

  heckman_df = tryCatch({
    data.frame(
      variable     = names(res$heckman$coefficients),
      Estimate     = as.numeric(res$heckman$coefficients),
      lower_ci     = res$heckman$ci[,1],
      upper_ci     = res$heckman$ci[,2],
      model        = "selection",
      phi          = res$phi,
      sample_size  = res$sample_size,
      replicate    = res$replicate,
      stringsAsFactors = FALSE
    )
  }, error = function(e) {
    data.frame(
      variable     = names(res$heckman$coefficients),
      Estimate     = as.numeric(res$heckman$coefficients),
```

```
      lower_ci     = rep(NA, length(res$heckman$coefficients)),
      upper_ci     = rep(NA, length(res$heckman$coefficients)),
      model        = "selection",
      phi          = res$phi,
      sample_size  = res$sample_size,
      replicate    = res$replicate,
      stringsAsFactors = FALSE
    )
  })

  # Pattern Mixture Selection model summary
  pattern_mixture_df = tryCatch({
    data.frame(
      variable     = names(res$pattern_mixture$coefficients),
      Estimate     = as.numeric(res$pattern_mixture$coefficients),
      lower_ci     = res$pattern_mixture$ci[,1],
      upper_ci     = res$pattern_mixture$ci[,2],
      model        = "pattern mixture",
      phi          = res$phi,
      sample_size  = res$sample_size,
      replicate    = res$replicate,
      stringsAsFactors = FALSE
    )
  }, error = function(e) {
    data.frame(
      variable     = names(res$pattern_mixture$coefficients),
      Estimate     = as.numeric(res$pattern_mixture$coefficients),
      lower_ci     = rep(NA, length(res$pattern_mixture$coefficients)),
      upper_ci     = rep(NA, length(res$pattern_mixture$coefficients)),
      model        = "pattern mixture",
      phi          = res$phi,
      sample_size  = res$sample_size,
      replicate    = res$replicate,
      stringsAsFactors = FALSE
    )
  })

  rbind(naive_df, weighted_df, heckman_df, pattern_mixture_df)
}))

# True Coefficients
true_coefs = data.frame(
  variable   = c("age", "cholesterol", "hypertension", "smoking", "exercise"),
  true_value = c(-0.03, -0.02, -0.5, -0.7, 0.2)
)


results_df_plot = results_df %>%
  left_join(true_coefs, by = "variable") %>%
  mutate(bias = Estimate - true_value)

## Aggregate Results Across Replicates
summary_df = results_df_plot %>%
  group_by(model, phi, variable, sample_size) %>%
  summarize(
    avg_Estimate = mean(Estimate, na.rm = TRUE),
    avg_bias     = mean(bias, na.rm = TRUE),
    mse          = mean(bias^2, na.rm = TRUE),
    .groups      = "drop"
```

```
  )


summary_df = summary_df %>% drop_na(phi, avg_bias, mse)

## Average Bias Plot
bias_plot = ggplot(summary_df, aes(x = phi, y = avg_bias, color = model)) +
  geom_line() +
  geom_point() +
  facet_grid(variable ~ sample_size, scales = "free_y") +
  labs(
    title = "Average Bias by MNAR Strength and Sample Size",
    x = "MNAR Strength (phi)",
    y = "Average Bias (Estimate - True Value)"
  ) +
  theme_minimal()

print(bias_plot)

## MSE Plot MSE
mse_plot = ggplot(summary_df, aes(x = phi, y = mse, color = model)) +
  geom_line() +
  geom_point() +
  facet_grid(variable ~ sample_size, scales = "free_y") +
  labs(
    title = "Mean Squared Error (MSE) by MNAR Strength and Sample Size",
    x = "MNAR Strength (phi)",
    y = "Mean Squared Error"
  ) +
  theme_minimal()

print(mse_plot)

## Average Coefficient Estimates Plot
coef_plot = ggplot(summary_df, aes(x = phi, y = avg_Estimate, color = model)) +
  geom_line() +
  geom_point() +
  facet_grid(variable ~ sample_size, scales = "free_y") +
  geom_hline(
    data = true_coefs,
    aes(yintercept = true_value),
    color = "black",
    linetype = "dashed"
  ) +
  labs(
    title = "Average Coefficient Estimates by MNAR Strength and Sample Size",
    x = "MNAR Strength (phi)",
    y = "Average Coefficient Estimate"
  ) +
  theme_minimal()

print(coef_plot)

## NATIVE AND WEIGHTED COMPARISON

results_df = do.call(rbind, lapply(results_list, function(res) {
  # Na ve model summary
  naive_df = tryCatch({
    data.frame(
```

```
      variable    = names(res$naive$coefficients),
      Estimate    = as.numeric(res$naive$coefficients),
      lower_ci    = res$naive$ci[,1],
      upper_ci    = res$naive$ci[,2],
      model       = "naive",
      phi         = res$phi,
      sample_size = res$sample_size,
      replicate   = res$replicate,
      stringsAsFactors = FALSE
    )
  }, error = function(e) {
    data.frame(
      variable    = names(res$naive$coefficients),
      Estimate    = as.numeric(res$naive$coefficients),
      lower_ci    = rep(NA, length(res$naive$coefficients)),
      upper_ci    = rep(NA, length(res$naive$coefficients)),
      model       = "naive",
      phi         = res$phi,
      sample_size = res$sample_size,
      replicate   = res$replicate,
      stringsAsFactors = FALSE
    )
  })

  # Weighted model summary
  weighted_df = tryCatch({
    data.frame(
      variable    = names(res$weighted$coefficients),
      Estimate    = as.numeric(res$weighted$coefficients),
      lower_ci    = res$weighted$ci[,1],
      upper_ci    = res$weighted$ci[,2],
      model       = "weighted",
      phi         = res$phi,
      sample_size = res$sample_size,
      replicate   = res$replicate,
      stringsAsFactors = FALSE
    )
  }, error = function(e) {
    data.frame(
      variable    = names(res$weighted$coefficients),
      Estimate    = as.numeric(res$weighted$coefficients),
      lower_ci    = rep(NA, length(res$weighted$coefficients)),
      upper_ci    = rep(NA, length(res$weighted$coefficients)),
      model       = "weighted",
      phi         = res$phi,
      sample_size = res$sample_size,
      replicate   = res$replicate,
      stringsAsFactors = FALSE
    )
  })


  rbind(naive_df, weighted_df)
}))

### True Coefficients
true_coefs = data.frame(
  variable   = c("age", "cholesterol", "hypertension", "smoking", "exercise"),
  true_value = c(-0.03, -0.02, -0.5, -0.7, 0.2)
```

```
)


results_df_plot = results_df %>%
  left_join(true_coefs, by = "variable") %>%
  mutate(bias = Estimate - true_value)


summary_df = results_df_plot %>%
  group_by(model, phi, variable, sample_size) %>%
  summarize(
    avg_Estimate = mean(Estimate, na.rm = TRUE),
    avg_bias     = mean(bias, na.rm = TRUE),
    mse          = mean(bias^2, na.rm = TRUE),
    .groups      = "drop"
  )


summary_df = summary_df %>% drop_na(phi, avg_bias, mse)

## Average Bias Plot
bias_plot = ggplot(summary_df, aes(x = phi, y = avg_bias, color = model)) +
  geom_line() +
  geom_point() +
  facet_grid(variable ~ sample_size, scales = "free_y") +
  labs(
    title = "Average Bias of Naive vs Weighted Models by MNAR Strength and Sample
        Size",
    x = "MNAR Strength (phi)",
    y = "Average Bias (Estimate - True Value)"
  ) +
  theme_minimal()

print(bias_plot)

## MSE Plot
mse_plot = ggplot(summary_df, aes(x = phi, y = mse, color = model)) +
  geom_line() +
  geom_point() +
  facet_grid(variable ~ sample_size, scales = "free_y") +
  labs(
    title = "Mean Squared Error (MSE) of Naive vs Weighted Models by MNAR Strength
        and Sample Size",
    x = "MNAR Strength (phi)",
    y = "Mean Squared Error"
  ) +
  theme_minimal()

print(mse_plot)

### Average Coefficient Estimates Plot
coef_plot = ggplot(summary_df, aes(x = phi, y = avg_Estimate, color = model)) +
  geom_line() +
  geom_point() +
  facet_grid(variable ~ sample_size, scales = "free_y") +
  geom_hline(
    data = true_coefs,
    aes(yintercept = true_value),
    color = "black",
```

```
    linetype = "dashed"
  ) +
  labs(
    title = "Average Coefficient Estimates of Naive vs Weighted Models by MNAR Strength and Sample Size",
    x = "MNAR Strength (phi)",
    y = "Average Coefficient Estimate"
  ) +
  theme_minimal()

print(coef_plot)

## AIC comparison
aic_list = list(naive = numeric(length(results_list)), weighted = numeric(length(results_list)))


for (i in seq_along(results_list)) {
  # Extract AIC for naive model
  if (!is.null(results_list[[i]]$naive_model)) {
    aic_list$naive[i] = tryCatch(
      AIC(results_list[[i]]$naive_model),
      error = function(e) {
        warning(sprintf("AIC failed for naive model in iteration %d: %s", i, e$message))
        return(NA)
      }
    )
  }


  if (!is.null(results_list[[i]]$weighted_model)) {
    aic_list$weighted[i] = tryCatch(
      AIC(results_list[[i]]$weighted_model),
      error = function(e) {
        warning(sprintf("AIC failed for weighted model in iteration %d: %s", i, e$message))
        return(NA)
      }
    )
  }
}

avg_aic = lapply(aic_list, mean, na.rm = TRUE)
print("Average AIC for Naive and Weighted Models:")
print(avg_aic)

aic_df = do.call(rbind, lapply(seq_along(results_list), function(i) {
  if (is.na(aic_list$naive[i])) return(NULL)
  data.frame(
    phi = results_list[[i]]$phi,
    sample_size = results_list[[i]]$sample_size,
    naive_aic = aic_list$naive[i],
    weighted_aic = aic_list$weighted[i]
  )
}))

aic_df_long = aic_df %>%
```

```r
  tidyr::pivot_longer(cols = c(naive_aic, weighted_aic), names_to = "model",
      values_to = "aic") %>%
  mutate(model = gsub("_aic", "", model))

# AIC plot
aic_plot = ggplot(aic_df_long, aes(x = phi, y = aic, color = model)) +
  geom_point() +
  geom_line() +
  facet_wrap(~ sample_size) +
  labs(
    title = "AIC for Naive vs Weighted Cox Models",
    x = "MNAR Strength (phi)",
    y = "AIC",
    color = "Model"
  ) +
  theme_minimal()

print(aic_plot)

### STANDARD ERROR

se_df = do.call(rbind, lapply(results_list, function(res) {
  naive_se = sqrt(diag(vcov(res$naive_model)))
  naive_se = naive_se[names(naive_se) != "Log(scale)"]


  weighted_se = sqrt(diag(vcov(res$weighted_model)))
  weighted_se = weighted_se[names(weighted_se) != "Log(scale)"]

  selection_se = sqrt(diag(vcov(res$heckman_model)))
  selection_se = selection_se[names(selection_se) != "IMR"]


  pattern_mixture_se = res$pattern_mixture$standard_errors

  df_naive = data.frame(
    phi = res$phi,
    sample_size = res$sample_size,
    model = "Naive",
    variable = names(naive_se),
    se = as.numeric(naive_se),
    stringsAsFactors = FALSE
  )

  df_weighted = data.frame(
    phi = res$phi,
    sample_size = res$sample_size,
    model = "Weighted",
    variable = names(weighted_se),
    se = as.numeric(weighted_se),
    stringsAsFactors = FALSE
  )

  df_selection = data.frame(
    phi = res$phi,
    sample_size = res$sample_size,
    model = "Selection",
    variable = names(selection_se),
    se = as.numeric(selection_se),
```

```
    stringsAsFactors = FALSE
  )

  df_pattern = data.frame(
    phi = res$phi,
    sample_size = res$sample_size,
    model = "Pattern Mixture",
    variable = names(pattern_mixture_se),
    se = as.numeric(pattern_mixture_se),
    stringsAsFactors = FALSE
  )

  rbind(df_naive, df_weighted, df_selection, df_pattern)
}))


ggplot(se_df, aes(x = phi, y = se, color = model, group = model)) +
  geom_line() +
  geom_point() +
  facet_grid(variable ~ sample_size, scales = "free_y") +
  labs(
    title = "Standard Errors by MNAR Strength and Sample Size",
    x = "MNAR Strength (phi)",
    y = "Standard Error",
    color = "Model"
  ) +
  theme_minimal()
```