Lecture 17: MCMC and Missing Data

CS109A: Introduction to Data Science

Harvard University

- **Course:** CS109A: Introduction to Data Science
- **Lecture:** Lecture 17
- **Instructors:** Pavlos Protopapas, Kevin Rader, Chris Gumb
- **Topics:** MCMC (Markov Chain Monte Carlo), Metropolis-Hastings Algorithm, Missing Data, Imputation

Lecture Overview

This lecture covers two important but distinct topics in data science:

1. **MCMC (Markov Chain Monte Carlo):** When Bayesian posterior distributions don't have nice closed forms (like in logistic regression), we need numerical methods to sample from them. MCMC provides a powerful framework for this.
2. **Missing Data:** Real-world datasets often have missing values. How you handle missing data can dramatically affect your results. We'll learn principled approaches beyond simply dropping rows.

Key Takeaway: These methods help us deal with the messy reality of statistical modeling—complex posteriors that can't be solved analytically, and datasets that aren't complete.

Contents

1 Review: Why Do We Need MCMC?

1.1 The Problem with Non-Conjugate Posteriors

In previous lectures, we discussed Bayesian inference and the concept of **conjugacy**:

- **Normal-Normal:** Normal prior on mean μ with Normal likelihood gives Normal posterior
- **Gamma-Normal:** Gamma prior on precision $(1/\sigma^2)$ gives Gamma posterior
- **Beta-Binomial:** Beta prior on probability p with Binomial likelihood gives Beta posterior

Key Information

Why Conjugacy is Nice:
When the posterior has a known distribution family, we can:

- Write down the posterior analytically
- Easily sample from it using standard functions
- Compute means, variances, credible intervals directly

1.2 What Happens in Logistic Regression?

When we tried to apply Bayesian methods to logistic regression, things fell apart:

1. Our unknown parameters are α (intercept) and β (slope)
2. These are linked to probability through the logistic function:

$$p = \frac{e^{\alpha+\beta x}}{1+e^{\alpha+\beta x}}$$

3. We put Normal priors on α and β
4. The posterior is no longer a named distribution!

Warning

The Core Problem:
The logistic function is **nonlinear**, and this nonlinearity destroys the nice conjugate relationship. The Beta-Binomial-Beta pattern doesn't work when we transform through $\frac{e^x}{1+e^x}$.
The posterior is some complicated function that we can write down but can't identify as any standard distribution.

1.3 What Can We Still Do?

Even though we can't name the posterior distribution, we can:

- **Evaluate its height:** Given any (α, β), we can compute the posterior density (up to a normalizing constant)
- **Compare relative probabilities:** We can tell which (α, β) pairs are more likely

The question becomes: *How do we explore and summarize a distribution we can evaluate but can't*

sample from directly?

Key Summary

Key Insight:

The posterior is *proportional to* prior $\times$ likelihood:

$$\pi(\alpha, \beta|\text{data}) \propto \pi(\alpha) \cdot \pi(\beta) \cdot L(\text{data}|\alpha, \beta)$$

We know the right-hand side. We just don't know what distribution it forms.

2 Monte Carlo Methods: The Big Picture

2.1 What is Monte Carlo?

Definition:

Monte Carlo Methods **Monte Carlo methods** are computational algorithms that use random sampling to obtain numerical results. The idea is simple: if you want to understand a distribution, generate many random samples from it and use those samples to estimate properties like the mean, variance, or credible intervals.

Think of it like this: instead of solving a complex integral analytically, we:

1. Generate thousands of random samples from the distribution
2. Use the empirical distribution of samples to estimate whatever we need

Example:

Estimating π with Monte Carlo Imagine a square with a circle inscribed inside it:

- Square has side length 2 (area = 4)
- Circle has radius 1 (area = π)

To estimate π:

1. Randomly throw darts at the square (uniform random points)
2. Count how many land inside the circle
3. Ratio $\approx$ Circle area / Square area = $\pi/4$

If 78.5% of 10,000 darts land in the circle, we estimate $\pi \approx 4 \times 0.785 = 3.14$

2.2 Where Have We Seen This Before?

Monte Carlo principles appear throughout statistics:

- **Bootstrapping:** Resample data to estimate sampling distribution of statistics
- **Permutation testing:** Randomly permute labels to create null distribution
- **Cross-validation:** Randomly split data to estimate model performance

All of these use repeated random sampling to empirically estimate something we can't compute analytically.

3 Rejection Sampling

3.1 The Basic Idea

Rejection sampling is a technique for sampling from a **target distribution** when we can only easily sample from a different **proposal distribution**.

Definition:

Rejection Sampling **Goal:** Sample from target distribution $f(x)$ (e.g., our posterior)
Tool: We can sample from proposal distribution $g(x)$ (e.g., a Normal)
Requirement: $f(x) \leq M \cdot g(x)$ for all x and some constant M

The analogy: *We want to sample from the unit circle, but we can only sample from the unit square. So we sample from the square and reject points outside the circle.*

3.2 The Cookie Monster Example

Imagine Cookie Monster is analyzing cookies from a bakery with four types: chocolate chip, oatmeal raisin, peanut butter, and sugar. Each type has different size distributions:

- Chocolate chip: centered around 8 cm diameter
- Oatmeal raisin: centered around 10 cm
- Peanut butter: centered around 12 cm
- Sugar: centered around 14 cm

When we look at the **marginal distribution** of all cookies combined, we get a complex multimodal distribution (possibly 3-4 peaks visible).

How do we sample from this weird distribution?

3.3 The Algorithm

1. Choose a proposal distribution $g(x)$ (e.g., a single Normal) that we can sample from
2. Scale it up by constant M so that $M \cdot g(x) \geq f(x)$ everywhere
3. Repeat:
   - Draw x from $g(x)$
   - Draw u from Uniform(0, 1)
   - If $u < \frac{f(x)}{M \cdot g(x)}$, accept x; otherwise reject

Key Information

Why This Works:
- When $f(x)$ is close to $M \cdot g(x)$: We accept most of the time
- When $f(x)$ is much smaller than $M \cdot g(x)$: We reject most of the time

The acceptance probability is exactly $\frac{f(x)}{M \cdot g(x)}$, which ensures we end up with samples distributed

according to $f(x)$.

3.4 Practical Example

Suppose:

- Target (black curve): Complex multimodal cookie size distribution
- Proposal (red curve): Single Normal, scaled up by $M = 11$

At $x = 10$ (center):

- Black curve height: 0.05
- Red curve height: 0.55
- Acceptance probability: $0.05/0.55 \approx 9\%$

At $x = 14$ (tail where black is higher relative to red):

- Black curve height: 0.04
- Red curve height: 0.045
- Acceptance probability: $0.04/0.045 \approx 89\%$

Warning

The Inefficiency Problem:
If $M = 11$, on average only about 1 in 11 proposals are accepted!
To get 10,000 samples from the target, we need to generate about 110,000 samples from the proposal.
This gets even worse in high dimensions.

3.5 Using the Samples

Once we have enough accepted samples, we can:

- Plot the histogram $\rightarrow$ Visual representation of posterior
- Compute the mean $\rightarrow$ Posterior mean estimate
- Find the 2.5th and 97.5th percentiles $\rightarrow$ 95% credible interval

Just like in bootstrapping, we use the **empirical distribution** of samples to estimate properties of the **theoretical distribution**.

3.6 Limitations of Rejection Sampling

1. **Computational waste:** Many proposals are rejected
2. **Curse of dimensionality:** In high dimensions (e.g., logistic regression with 20 predictors = 21 parameters), finding a good proposal distribution is nearly impossible
3. **Scaling constant M:** Needs to be large enough to cover the target everywhere, leading to low acceptance rates

Key Summary

Rejection sampling is conceptually simple and works well in low dimensions, but it becomes impractical for complex, high-dimensional posteriors. We need a better approach: **MCMC**.

4 Markov Chain Monte Carlo (MCMC)

4.1 What is MCMC?

Definition:

MCMC **Markov Chain Monte Carlo (MCMC)** is a class of algorithms that sample from a probability distribution by constructing a Markov chain that has the desired distribution as its equilibrium distribution.

Let's break this down:

- **Monte Carlo:** Using random sampling
- **Markov Chain:** A sequence where each step depends only on the previous step

4.2 The Markovian Property

Definition:

Markov Property (Memorylessness) A process has the **Markov property** if the probability of being in a future state depends *only* on the current state, not on how we got there.

$$P(\theta_{t+1}|\theta_t, \theta_{t-1}, \ldots, \theta_0) = P(\theta_{t+1}|\theta_t)$$

Example:

Real-World Markov Chains **Weather:**

- Tomorrow's weather depends mainly on today's weather
- If it's raining today, there's a higher chance of rain tomorrow
- We don't need to know what happened last week

Stock Prices:

- Tomorrow's price depends on today's price
- The path that got us to today's price doesn't matter
- "Where you are" is what matters, not "how you got there"

4.3 The Key Insight of MCMC

Instead of sampling independently from the distribution (like rejection sampling), MCMC creates a **random walk** through the parameter space.

Imagine a robot walking around a landscape where height represents probability:

- Higher terrain = Higher posterior probability
- The robot tends to move uphill (toward higher probability)
- Sometimes it moves downhill (but less often)
- Over time, the robot spends more time in high-probability regions

Key Information

Why MCMC Works:

If we construct the random walk correctly, the **long-run frequency** of visits to any region will be proportional to the posterior probability of that region.

After enough steps, the sequence of positions forms a sample from the posterior distribution!

5 The Metropolis-Hastings Algorithm

5.1 Algorithm Overview

The **Metropolis-Hastings algorithm** is the most common MCMC method:

1. Start at some initial parameter value θ_0
2. At each step t:
   - Propose a new value θ^* based on current position θ_t
   - Decide whether to accept or reject the proposal
   - If accept: $\theta_{t+1} = \theta^*$
   - If reject: $\theta_{t+1} = \theta_t$ (stay in place)
3. Repeat many times

5.2 The Proposal Distribution

The **proposal distribution** $q(\theta^*|\theta_t)$ determines how we suggest new values:

Common choice: **Random walk proposal**

$$\theta^* = \theta_t + \epsilon, \quad \epsilon \sim \text{Normal}(0, \sigma^2)$$

This means: "Take a random step from current position"

- Small σ: Small steps, slow exploration
- Large σ: Big steps, might miss important regions

5.3 The Acceptance Probability

Here's the key formula—the probability of accepting a proposed move:

$$\alpha = \min\left(1, \frac{f(\theta^*)}{f(\theta_t)} \cdot \frac{q(\theta_t|\theta^*)}{q(\theta^*|\theta_t)}\right) \tag{1}$$

Where:

- $f(\theta)$ is the target distribution (posterior, up to normalizing constant)
- $q(\theta^*|\theta_t)$ is the proposal distribution

Key Information

Intuition Behind the Formula:
Ratio $\frac{f(\theta^*)}{f(\theta_t)}$:

- If θ^* has higher posterior probability → ratio > 1
- We accept with probability 1 (always move uphill!)
- If θ^* has lower probability → ratio < 1

- We accept with that probability (sometimes move downhill)

The proposal correction term $\frac{q(\theta_t|\theta^*)}{q(\theta^*|\theta_t)}$:

- For symmetric proposals (like random walk), this equals 1
- Needed for asymmetric proposals to ensure proper sampling

5.4 Step-by-Step Example

Suppose we're at θ_t where $f(\theta_t) = 0.5$

We propose θ^* where $f(\theta^*) = 0.3$ (downhill move)

Acceptance probability (with symmetric proposal):

$$\alpha = \min\left(1, \frac{0.3}{0.5}\right) = 0.6$$

We generate $u \sim \text{Uniform}(0, 1)$:

- If $u < 0.6$: Accept, move to θ^*
- If $u \geq 0.6$: Reject, stay at θ_t

Key Summary

Key Properties of Metropolis-Hastings:

1. **Uphill moves:** Always accepted
2. **Downhill moves:** Sometimes accepted (proportional to ratio)
3. **Result:** Spends more time in high-probability regions
4. **Doesn't require normalizing constant:** Only needs ratios!

6 Practical MCMC: The Skittles Example

6.1 The Problem Setup

A candy company wants to introduce a new Skittles flavor: Mango. They need to determine the optimal amount of "mango essence" ingredient.

Data collection:

- Different dosages (amounts of mango essence) are tested
- Multiple taste testers try each dosage
- Each tester says "Yes, I love it" or "No"

This is a classic **dose-response** problem!

6.2 The Model

- **Predictor x:** Amount of mango essence (mg)
- **Response y:** Number of people who love it out of n testers
- **Distribution:** $Y \sim \text{Binomial}(n, p(x))$

We link p to x through logistic regression:

$$\log\left(\frac{p}{1-p}\right) = \alpha + \beta x$$

Or equivalently:

$$p = \frac{e^{\alpha+\beta x}}{1+e^{\alpha+\beta x}}$$

6.3 Bayesian Setup

Unknown parameters: α (intercept) and β (slope)

Priors:

$$\alpha \sim \text{Normal}(0, 100^2)$$
$$\beta \sim \text{Normal}(0, 100^2)$$

Why Normal(0, 100)?

- Centered at 0: No prior assumption about direction of effect
- Large variance (100): Very non-informative, letting data speak
- On log-odds scale, α and β can be any real number

Posterior:

$$\pi(\alpha, \beta|\text{data}) \propto \pi(\alpha) \cdot \pi(\beta) \cdot \prod_{i=1}^{n} \binom{n_i}{y_i} p_i^{y_i}(1-p_i)^{n_i-y_i}$$

This is a complicated function—not a named distribution!

6.4 Using PyMC for MCMC

```python
import pymc as pm
import numpy as np

# Data
flavoring = np.array([1.88, 1.86, 1.84, 1.82, ...])  # Dosages
y = np.array([60, 61, 58, 56, ...])  # Number who loved it
n = np.array([60, 62, 60, 58, ...])  # Total testers

with pm.Model() as skittles_model:
    # Priors
    alpha = pm.Normal('alpha', mu=0, sigma=100)
    beta = pm.Normal('beta', mu=0, sigma=100)

    # Logistic transformation
    p = pm.math.invlogit(alpha + beta * flavoring)

    # Likelihood
    y_obs = pm.Binomial('y_obs', n=n, p=p, observed=y)

    # MCMC Sampling
    trace = pm.sample(2000, tune=2000, return_inferencedata=True)
```

Key parameters:

- `tune=2000`: Burn-in period (samples to discard)
- `2000`: Number of samples to keep after burn-in

6.5 Interpreting the Results

After running MCMC, we get a **trace**—a sequence of sampled (α, β) pairs.

Trace plots show the sampled values over time:

- Should look like "fuzzy caterpillars" or "hairy worms"
- No trends or patterns
- Bouncing randomly around a stable mean

Warning

Signs of Problems in Trace Plots:

- Gradual drift upward or downward
- Getting stuck in one place for many iterations
- Different chains showing different patterns

These indicate the chain hasn't converged to the posterior!

Posterior histograms show the distribution of samples:

- For β: Values range from about 25 to 42

- All positive! This tells us more mango essence → more people love it
- Posterior mean: around 33-34
- 95% credible interval: approximately [28, 40]

7 MCMC Diagnostics

7.1 The Burn-in Period

Definition:

Burn-in (Warm-up) The **burn-in** period is the initial portion of the MCMC chain that is discarded before collecting samples. During burn-in, the chain is "warming up" and moving from its arbitrary starting point toward the high-probability region.

Why is burn-in necessary?

- We start the chain at some arbitrary initial value
- If we start in a low-probability region, early samples don't represent the posterior
- We wait until the chain "finds" the high-probability region

7.2 The R-hat Statistic ($\hat{R}$)

Definition:

R-hat ($\hat{R}$) **R-hat** (also called the potential scale reduction factor) measures whether multiple MCMC chains have converged to the same distribution.

$$\hat{R} \approx 1.0 \quad \text{indicates convergence}$$

How it works:

1. Run multiple independent chains (typically 4)
2. Compare the variance *within* each chain to variance *between* chains
3. If chains have converged, these should be similar ($\hat{R} \approx 1$)
4. If chains are exploring different regions, between-chain variance is larger ($\hat{R} > 1$)

Important:

Convergence Rules

- $\hat{R} < 1.01$: Excellent convergence
- $\hat{R} < 1.05$: Generally acceptable
- $\hat{R} > 1.1$: **Serious problem!** Do not trust results!

If $\hat{R} > 1.1$, try:

1. Increase burn-in period
2. Run chains longer
3. Try different initial values
4. Reparameterize the model

7.3 Effective Sample Size (ESS)

Because MCMC samples are **correlated** (each depends on the previous), they don't provide as much information as independent samples.

Definition:

Effective Sample Size The **effective sample size (ESS)** is the equivalent number of independent samples that would provide the same information as the correlated MCMC samples.
If you have 2000 MCMC samples but ESS = 500, your samples are as informative as 500 independent samples.

The MCMC standard error for the mean is:

$$\text{MCSE} = \frac{\text{Posterior SD}}{\sqrt{\text{ESS}}}$$

Not $\sqrt{n}$ where n is the number of MCMC samples!

7.4 Summary Statistics from PyMC

Typical output includes:

- **mean**: Posterior mean estimate
- **sd**: Posterior standard deviation
- **hdi_3%**, **hdi_97%**: 94% credible interval bounds
- **mcse_mean**: Monte Carlo standard error for the mean
- **ess_bulk**: Effective sample size
- **r_hat**: Convergence diagnostic

Key Summary

MCMC Summary:

1. MCMC constructs a random walk through parameter space
2. After burn-in, samples represent the posterior distribution
3. Always check diagnostics: trace plots, $\hat{R}$, ESS
4. Use posterior samples to compute means, credible intervals, predictions

8 Missing Data: Introduction

8.1 What is Missing Data?

In real-world datasets, values are often missing:

- Survey respondents skip questions
- Sensors fail to record measurements
- Data entry errors leave blanks
- Information is not collected for some subjects

How does Python handle missing values?

- Pandas represents missing as `NaN` (Not a Number) or `NA`
- Scikit-learn **throws errors** if given data with `NaN`
- You must handle missing data before modeling!

8.2 Common (But Problematic) Approaches

Option 1: Drop rows with missing values

```
df_clean = df.dropna()  # Remove rows with any NaN
```

Option 2: Drop columns with missing values

```
df_clean = df.dropna(axis=1)  # Remove columns with any NaN
```

Option 3: Fill with mean/median

```
df['column'] = df['column'].fillna(df['column'].mean())
```

Warning

Problems with Simple Approaches:

Dropping rows:

- Loses valuable data
- Can introduce **bias** if missingness is related to the outcome
- Reduces sample size and statistical power

Mean imputation:

- **Underestimates variance** (makes distribution narrower)
- Distorts relationships between variables
- Doesn't account for uncertainty in imputed values

9 Types of Missing Data

Understanding *why* data is missing is crucial for choosing the right approach.

9.1 Missing Completely at Random (MCAR)

Definition:

MCAR Data is **Missing Completely at Random** if the probability of being missing is unrelated to:

- The missing value itself
- Any other observed variables

Missingness is purely random, like someone randomly punching holes in your spreadsheet.

Example: A survey is accidentally not sent to 5% of randomly selected participants.

Implication: Complete case analysis (dropping rows) gives unbiased estimates, just with reduced sample size.

9.2 Missing at Random (MAR)

Definition:

MAR Data is **Missing at Random** if the probability of being missing can be fully explained by other *observed* variables (but not by the missing value itself).

Example: In a depression study, younger people are less likely to report their income. The missingness is related to age (which we observe), not to income itself.

Implication: We can use observed variables to model the missingness and correct for bias.

9.3 Missing Not at Random (MNAR)

Definition:

MNAR Data is **Missing Not at Random** if the probability of being missing depends on the unobserved value itself or on unobserved variables.

Example: People with very high incomes are less likely to report their income *because* it's high.

Implication: This is the hardest case. No statistical method can fully correct for the bias without strong assumptions.

Important:

The Fundamental Problem We can never know for certain which type of missingness we have. The classification depends on information we don't observe. This is why missing data handling requires careful thought and often sensitivity analysis.

10 Better Approaches to Missing Data

10.1 The Indicator Variable Method

A simple but often effective approach:

1. Create a new variable indicating whether the original value was missing
2. Impute a constant (like 0) for missing values
3. Include **both** the imputed variable and the indicator in your model

Example:

Indicator Method Original data:

ID	Income	Outcome
1	50000	1
2	NaN	0
3	75000	1
4	NaN	1

After transformation:

ID	Income_imputed	Income_missing	Outcome
1	50000	0	1
2	0	1	0
3	75000	0	1
4	0	1	1

Why this works:

- The imputed zeros aren't treated as real zeros
- The indicator variable captures the "missingness effect"
- The model can learn different relationships for missing vs. non-missing cases

```python
import pandas as pd
import numpy as np

def add_missing_indicator(df, column):
    """Add missing indicator and impute zeros"""
    # Create indicator
    df[f'{column}_missing'] = df[column].isna().astype(int)
    # Impute with 0
    df[f'{column}_imputed'] = df[column].fillna(0)
    return df

# Usage
df = add_missing_indicator(df, 'income')
# Now use both 'income_imputed' and 'income_missing' in model
```

10.2 Categorical Variables

For categorical variables, treat "missing" as its own category:

Example:

Missing as a Category Original variable `color` with values: Red, Blue, Green, NaN
After transformation: Red, Blue, Green, Missing
This is equivalent to one-hot encoding with three original categories plus one missing category.

10.3 Multiple Imputation (Advanced)

The gold standard for handling missing data:

1. Create multiple (5-20) imputed datasets, each with slightly different imputed values
2. Analyze each dataset separately
3. Combine results using special rules that account for imputation uncertainty

This is more complex but properly accounts for uncertainty about the missing values.

11 Key Takeaways

Key Summary

MCMC and Missing Data: Summary

MCMC:

- When posteriors don't have closed forms, MCMC lets us sample from them
- Metropolis-Hastings: Propose → Accept/Reject → Repeat
- Always check convergence: trace plots, $\hat{R} \approx 1.0$, adequate ESS
- Use samples to estimate means, credible intervals, predictions

Missing Data:

- Three types: MCAR, MAR, MNAR (we usually don't know which)
- Simple dropping or mean imputation can bias results
- Better: Indicator variable method (impute + flag)
- Best: Multiple imputation (accounts for uncertainty)

The Common Theme: Both topics deal with **uncertainty**—uncertainty in posterior distributions (MCMC) and uncertainty about missing values (imputation). Good data science acknowledges and properly accounts for uncertainty.

Table 1: *MCMC vs. Rejection Sampling Comparison*

Aspect	Rejection Sampling	MCMC
Sample independence	Independent samples	Correlated samples
Efficiency in high-D	Very poor	Much better
Requires	Envelope distribution $M \cdot g(x)$	Proposal distribution q
Acceptance rate	Can be very low	Tunable (aim for 20-50%)
Burn-in needed	No	Yes
Convergence diagnostics	Not applicable	Essential ($\hat{R}$, ESS)

Table 2: *Missing Data Handling Methods*

Method	Pros	Cons
Drop rows (listwise deletion)	Simple, unbiased if MCAR	Loses data, biased if not MCAR
Mean imputation	Simple, preserves sample size	Underestimates variance, biased
Indicator variable method	Captures missingness pattern	Assumes missingness is informative
Multiple imputation	Proper uncertainty quantification	Complex, computationally intensive

12 Python Code Reference

12.1 MCMC with PyMC

```
import pymc as pm
import arviz as az  # For diagnostics
```

```
3
4  # Define model
5  with pm.Model() as model:
6      # Priors
7      alpha = pm.Normal('alpha', mu=0, sigma=10)
8      beta = pm.Normal('beta', mu=0, sigma=10)
9
10     # Likelihood (logistic regression example)
11     p = pm.math.invlogit(alpha + beta * X)
12     y_obs = pm.Binomial('y_obs', n=n, p=p, observed=y)
13
14     # Sample
15     trace = pm.sample(2000, tune=2000, cores=4,
16                       return_inferencedata=True)
17
18 # Diagnostics
19 az.plot_trace(trace)  # Trace plots
20 az.summary(trace)     # Summary statistics including r_hat
```

12.2 Missing Data Handling

```
1  import pandas as pd
2  import numpy as np
3  from sklearn.impute import SimpleImputer
4
5  # Method 1: Indicator variable approach
6  def handle_missing_with_indicator(df, columns):
7      for col in columns:
8          df[f'{col}_missing'] = df[col].isna().astype(int)
9          df[col] = df[col].fillna(0)  # or fillna(df[col].mean())
10     return df
11
12 # Method 2: Using sklearn
13 imputer = SimpleImputer(strategy='mean')  # or 'median', 'most_frequent'
14 X_imputed = imputer.fit_transform(X)
15
16 # Check missingness pattern
17 print(df.isna().sum())  # Count NaN per column
18 print(df.isna().mean())  # Proportion NaN per column
```

13 Checklist for Understanding

☐ Can you explain why we need MCMC when the posterior isn't a named distribution?

☐ Can you describe the basic idea of Metropolis-Hastings (propose, accept/reject)?

☐ Do you understand what trace plots should look like for a converged chain?

☐ Can you interpret $\hat{R}$ and know when it indicates problems?

☐ Do you know the three types of missing data (MCAR, MAR, MNAR)?

☐ Can you implement the indicator variable method for handling missing data?

☐ Do you understand why simple mean imputation can be problematic?