Missing Value Imputation: Evaluating the Impact of Outcome Inclusion on Predictive Utility

Author: Hongda Yuan
Date: December 2025

---

a. Motivation

Missing data is ubiquitous in real-world datasets. When building predictive models, researchers must decide: **Should the outcome variable be included in the imputation process, especially when using deep learning methods?**

This question matters because:

1. **Deep Learning Methods:** Modern deep learning imputation methods (Autoencoders, GAIN) are increasingly popular, but there is limited guidance on whether these methods should include the outcome variable during training.
2. **Downstream Task Focus:** The primary goal of imputation in many applications is to produce imputed data that **maximizes performance on downstream prediction tasks** (e.g., predicting disease outcomes in clinical settings), not to perfectly recover missing values.

---

b. Project Description

Components

A comprehensive simulation framework for evaluating missing value imputation methods, focusing on comparing imputation strategies that include versus exclude the outcome variable. The framework includes:

1. **Data Generation:** Synthetic data with configurable complexity (interactions, nonlinear terms, splines)
2. **Missingness Application:** Six missingness patterns (MCAR, MAR, MNAR, etc.)
3. **Imputation Methods:** Eight methods (Complete Data, Mean, Single, MICE, MissForest, MLP, Autoencoder, GAIN), each tested with three variants (without outcome, with binary Y, with continuous Y_score)

4. **Utility-Based Evaluation:** Train downstream models (Logistic/Linear Regression) on imputed data, evaluate on complete test sets
5. **Analysis Tools:** Statistical tests and visualizations

Work Flow

The simulation follows: **Data Generation → Missingness Application → Imputation → Evaluation → Aggregation**

Key Innovation: Utility-based evaluation measures downstream prediction performance (Log Loss for binary Y, R^2 for continuous Y_score) rather than imputation accuracy, directly answering: "Does this imputation method produce data that leads to better predictions?"

Course Techniques Used

- **Software Engineering:** Object-oriented design, parallel processing (multiprocessing), configuration management (JSON), testing frameworks (pytest), profiling (cProfile)
- **Data Science Tools:** NumPy/Pandas, Scikit-learn, PyTorch, Matplotlib/Seaborn
- **High-Performance Computing:** SLURM job scheduling, resource allocation

---

c. Results or Demonstration

Simulation Configuration

Combined Analysis: 120 method-pattern combinations from two simulations:

- **CPU Simulation:** 100 runs, 14 methods (traditional + MLP)
- **GPU Simulation:** 50 runs, 6 methods (Autoencoder, GAIN)
- **Missingness Patterns:** 6 patterns (MCAR, MAR, MARType2Y, MARType2Score, MNAR, MARThreshold)

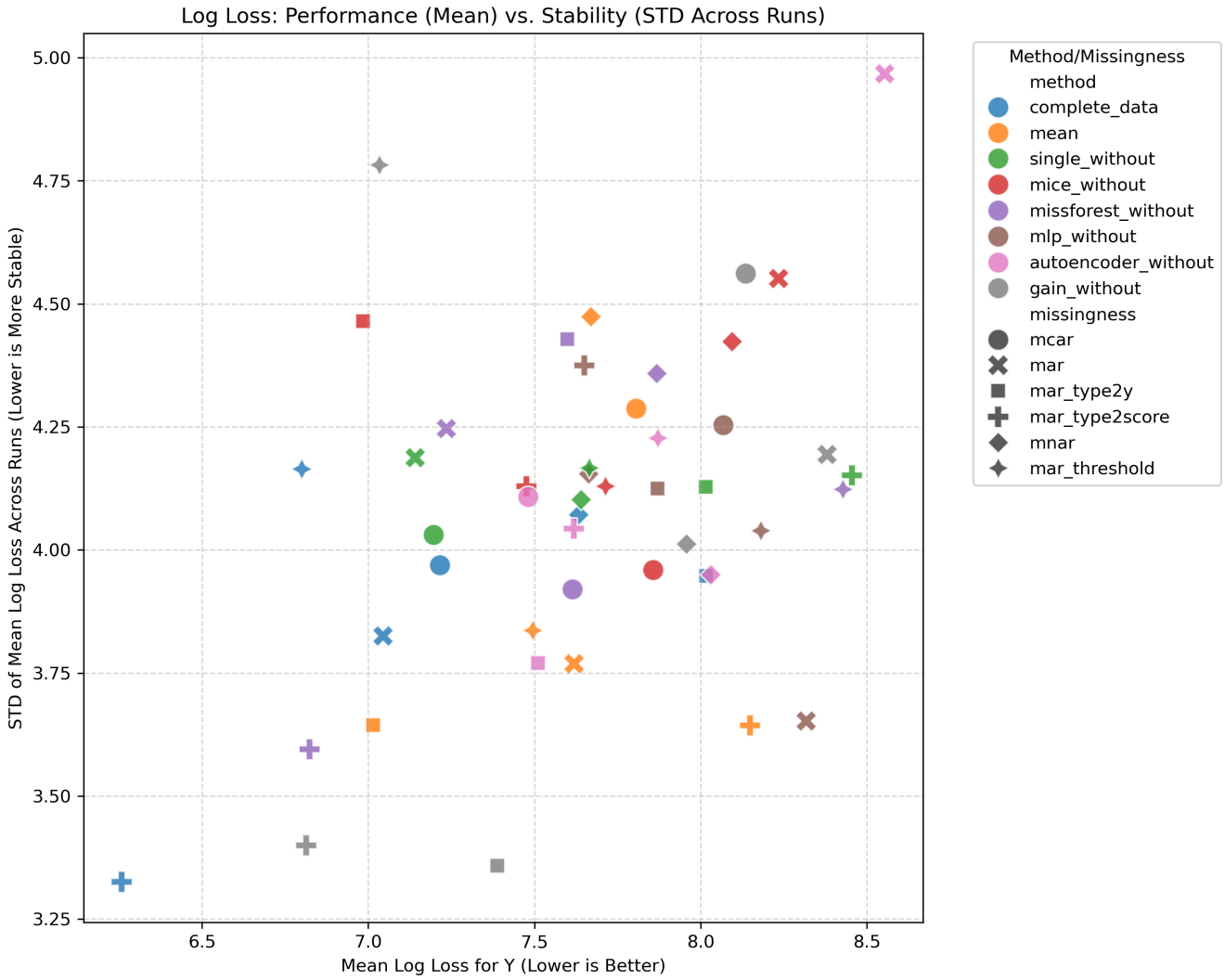
Key Findings

1. Statistical Tests

ANOVA tests show no significant difference at aggregate level (Log Loss: $F=1.19$, $p=0.279$; R^2: $F=1.00$, $p=0.463$), but method performance varies substantially by missingness pattern and outcome inclusion strategy.

Stability vs. Performance Trade-off

Figure: `y_log_loss_stability_plot.png`



Performance Summary (methods without outcome inclusion):

- **Best performance:** Complete Data (mean Log Loss: 7.16)
- **Deep learning methods:** Mean Log Loss 7.73, mean STD: 4.11
- **Traditional methods:** Mean Log Loss 7.74, mean STD: 4.15

Key Insight: Deep learning methods show **similar performance and stability** to traditional methods when outcome is excluded (7.73 vs. 7.74 Log Loss, 4.11 vs. 4.15 STD), contradicting the expectation that deep learning would show higher variability.

---

d. Lessons Learned

Challenges

1. **B-Spline Requirements:** Crashed with `ValueError: Need at least 8 knots for degree 3` for small n. **Solution:** Dynamically calculate knots: `num_knots = num_coeffs + degree + 1`.
2. **Performance Optimization:** Slow execution for large-scale simulations. **Solution:** Vectorization, caching, optimized aggregation, and multiprocessing. **Result:** 15-30% speedup.
3. **HPC Resource Allocation:** Initially considered GPU, but analysis showed CPU-bound workload. **Solution:** Requested 64 CPU cores instead of GPU. **Rationale:** 6/8 methods are CPU-only; parallelization across parameter combinations benefits from many cores.

Approach Changed

1. **Systematic Design:** Implemented full factorial design with JSON configuration, enabling systematic exploration of all method × missingness × outcome inclusion combinations.
2. **Parallel Execution:** Added multiprocessing and HPC integration (SLURM) for large-scale simulations, with resource-aware parallelization.
3. **Comprehensive Analysis:** Developed automated analysis pipeline with statistical tests (ANOVA), multiple visualizations (heatmaps, bar plots, stability plots), and uncertainty quantification.

Key Insight: The course emphasis on statistical rigor, reproducibility, and real-world applicability shaped the entire framework design.

Key Takeaways

1. **Numerical stability is critical**—edge cases in probabilities and logarithms can crash simulations.
2. **Test after changes**—regression testing ensures optimizations preserve correctness.
3. **Understand workload characteristics**—CPU vs. GPU allocation should match actual needs.
4. **Systematic design enables robust conclusions**—full factorial designs with proper Monte Carlo simulation.
5. **Utility over accuracy**—for prediction tasks, downstream performance matters more than imputation accuracy.