CS 598: Foundations of Data Curation - Final Project Report

Date: December 10, 2025
Curator: Hard Shah
Project: AI4I 2020 Predictive Maintenance Data Curation
GITHUB: https://github.com/Hard-22/HS-CS598-Project.git
ZENODO DOI: https://doi.org/10.5281/zenodo.17891383

1. Project Motivation and Context

The proliferation of "Industry 4.0" technologies has led to massive data generation in manufacturing, yet high-quality, documented datasets for predictive maintenance remain scarce. Machine failures, while rare, are critical events that can cause significant downtime and financial loss.

This project focuses on the curation of the AI4I 2020 Predictive Maintenance Dataset, a synthetic dataset reflecting real-world predictive maintenance scenarios. The primary use case is educational capability building and algorithm benchmarking. Specifically, the curated dataset is designed to help researchers and students understand the challenges of imbalanced classification in industrial settings—a scenario where failure events are vastly outnumbered by normal operations (96.6% vs 3.4%).

By curating this dataset, we aim to lower the barrier to entry for studying predictive maintenance, providing a clean, well-documented resource that highlights common data quality issues like class imbalance and feature correlation.

2. Dataset Profile

The AI4I 2020 Predictive Maintenance Dataset (located in `Artifacts & Workflow/data/`) consists of 10,000 observations of a simulated milling machine.

- Structure: Tabular data with 14 features.
- Size: 10,000 rows.

- Key Features:
- Process Variables: Air temperature [K], Process temperature [K], Rotational speed [rpm], Torque [Nm], Tool wear [min].
- Product Information: Type (Low/Medium/High quality variants).
- Target Variable: `Machine failure` (Binary: 0 or 1).
- Failure Modes: Five specific failure types (TWF, HDF, PWF, OSF, RNF) tracked as binary flags.
- Origin: Synthetically generated by Stephan Matzka (HTW Berlin) based on physics models.
- License: CC BY 4.0.

3. Data Curation Workflow

The curation workflow was implemented as a reproducible Python pipeline (`Artifacts & Workflow/main.py`), adhering to the DCC Curation Lifecycle Model. We moved beyond simple "cleaning" to a diverse set of activities:

3.1. Ingest & Validation

We implemented a robust loading mechanism (`scripts/load_data.py`) that doesn't just read the CSV but validates its structure against a defined schema. It asserts that all 14 columns exist and match expected data types. This ensures that any corruption in the raw file is caught immediately.

3.2. Data Transformation & Cleaning

Instead of manual edits, all transformations are scripted in `scripts/transform_data.py`:
- Outlier Detection: We used the Interquartile Range (IQR) method (1.5 * IQR) to scan continuous variables. Interestingly, we identified 418 outliers in `Rotational speed` and 69 in `Torque`. Crucially, we decided NOT to remove these. In the domain of failure prediction, extreme values are often the "signal" rather than "noise." Removing them would have sanitized the dataset to the point of uselessness for failure detection.

- Normalization: We applied Z-score normalization (StandardScaler) to the physics-based features. This scales values to have a mean of 0 and variance of 1, which is essential for many distance-based machine learning algorithms (like KNN or SVM) that students might use.

3.3. Provenance Tracking

We developed a custom `ProvenanceLogger` (`scripts/log_provenance.py`) that runs alongside the main script. It captures:
- Entities: The input file info (hash, size).
- Activities: Timestamps of when normalization and export occurred.
- Agents: The execution environment details.

This is exported as `Metadata & Documentation/metadata/provenance_record.json`, allowing for a W3C PROV-compliant audit trail.

4. Lifecycle Model Analysis

This project aligns closely with the DCC Curation Lifecycle Model:

- Conceptualize: We defined the need for a "clean" version of this dataset specifically for teaching imbalanced learning.
- Appraise & Select: The decision to retain outliers in Torque/Speed was a key appraisal step. We evaluated them as "significant" rather than "erroneous."
- Ingest: The data was validated upon entry.
- Preservation Action: We focused on Migration (converting to non-proprietary CSV/JSON formats in `Artifacts & Workflow/output`) and Technology Preservation (containerizing the environment via `requirements.txt`).
- Description: We enriched the data with DataCite-compliant metadata.
- Access: This self-contained submission package acts as the Dissemination Information Package (DIP).

5. Findings and Lessons Learned

- Class Imbalance is Severe: The 3.39% failure rate implies that accuracy is a misleading metric. A model that predicts "No Failure" for everyone achieves 96.6% accuracy but 0% utility. We documented this heavily in `docs/README.md` to guide future users toward F1-score and Precision/Recall.
- Metadata Standard Complexity: Mapping a simple tabular dataset to complex schemas like DataCite 4.4 required decision-making. For example, we treated the "Publisher" as the GitHub repository owner, which matches modern data sharing practices but differs from traditional library definitions.
- Provenance Overhead: Implementing granular provenance tracking added code complexity. However, it proved valuable when we needed to debug *why* the dataset size changed during development—we could trace it back to a (later reverted) drop-duplicates step in the logs.

6. Connection to Course Concepts

- Metadata (M8): We used the DataCite schema for dataset-level metadata and created a detailed Codebook (`Metadata & Documentation/output/data_dictionary.json`) for variable-level metadata. This fulfills the OAIS requirement for Representation Information—without it, a column like `Air temperature` is just a number; with it, we know it is in Kelvin.
- Quality (M6): We addressed quality via automated assertions. We ensured Completeness (100% rows have no nulls) and Consistency (types are correct). See `Metadata & Documentation/metadata/quality_report.json` for the evidence.
- Ethics (M2): While the data is synthetic, we successfully avoided the "black box" problem by documenting the known limitations and potential biases in the `README`. This ensures users know *what* they are using and its limits.
- Automation (M12): By using `main.py` instead of a Jupyter Notebook for the core logic, we ensured the process is production-ready and reproducible, avoiding the "hidden state" problems common in notebooks.

End of Report