Predicting Chicago Food Inspection Failures

Jingzhi Yang Lingyue Ji Nate Assefa

1. Data Description

1.1. Objective

Our research question is clear and focused: *Can we predict, at the moment an inspection occurs, whether a Chicago food facility will fail?* This question specifies both the prediction target, which is inspection failure, and the timing constraint, that only pre-inspection information may be used.

The City of Chicago Food Inspections dataset is well suited to this goal. It provides detailed facility characteristics, inspection metadata, location, and free-text violation notes. These fields capture operational risk factors and historical outcomes that are essential for modeling future failures.

Our research strategy is a supervised machine learning approach. We first clean and normalize the raw data, engineer features such as an ordinal risk score and calendar variables, and then train classification models using the cleaned training set while holding out a test set for final evaluation. This strategy matches the research question because it builds a predictive model while respecting the constraint that only information available before the inspection is included. The public documentation and data dictionary of the dataset serve as the primary codebook and reference for variable definitions.

1.2. Team Collaboration

Lingyue found the raw data, exported cleaned training and testing CSV files, and created the repository. Nate cleaned the data, checked schema, and performed visualization. Jingzhi managed the files, created the data dictionary, and wrote the report.

1.3. Variables

The dataset contains both original inspection fields and engineered features for modeling. Key original variables include:

- **inspection_id**: Unique identifier for each inspection event
- **inspection_date**: Calendar date of the inspection; used for time-based analysis
- **facility_type**: Establishment category (e.g., restaurant, grocery store, school) capturing operational risk
- **risk**: City-defined risk level with three categories: Risk 1 (High), Risk 2 (Medium), Risk 3 (Low)
- **inspection_type**: Reason for inspection (e.g., canvass, complaint, license, re-inspection)
- **zip**, **city**, **latitude**, **longitude**: Facility location variables for geographic analysis
- **results**: Official outcome string (pass, pass w/ conditions, fail, or out of business); defines the target label.
- **violations**: Free-text list of cited health-code violations, often semi-structured with numbered items

We also create engineered features to support modeling:

- **viol_count**: Count of numbered violations parsed from the free-text field
- **risk_ord**: Ordinal encoding of risk (High=3, Medium=2, Low=1)
- **ins_year**, **ins_month**, **ins_wday**: Year, month, and weekday extracted from the inspection date
- **target_fail**: Binary target variable; equals 1 if the inspection result is "fail," 0 otherwise

These variables together describe the facility, its risk context, the inspection circumstances, and outcomes. They form the basis for predicting inspection failure while avoiding data leakage by excluding violation text from the predictive model.

1.4. Data Cleaning

Each data split contains 1,000 inspections. After cleaning, the tables hold 18 columns, and about 19% of training records show a failed inspection. Facility type has about 41 categories, with restaurants most common. Inspection type has about 19 categories, and risk has three. Coordinates cluster tightly around Chicago (about 41.88° N, −87.67° W). Years range from 2010 to 2025 with even coverage. Failure rates rise with higher risk, though risk 2 shows a slightly higher rate than risk 3.

Cleaning follows a single reproducible pipeline. We drop duplicate inspection IDs and standardize text. City names convert to title case, and a new field flags Chicago locations. ZIP codes convert from floats to five-digit strings with leading zeros. Dates parse to `datetime` objects, and we derive year, month, and weekday. Result strings normalize to a compact set ("pass," "pass w/ conditions," "fail," and similar). We encode the risk field to an ordinal integer. Violation notes become a numeric count using a layered parsing strategy. Latitude and longitude outside conservative Chicago bounds become missing.

1.5. Challenges

The first dataset we considered was from Kaggle and was eight years ago. After discussion, we switched to the current dataset, which is more up-to-date.

Text fields contained inconsistent casing and spelling, so we standardized them. Some cells contained the string "nan," which we converted to missing. ZIP codes required conversion from float to string. Violation text is semi-structured, so we used a conservative counting method. The test set includes facility types not present in training, so we map unseen categories to an "Other" bucket and use encoders that tolerate unknown levels.

2. Pre-Analysis Plan

2.1. Method Overview

We treat this project as a supervised binary classification task. The goal is to predict whether a food inspection will *fail* (`target_fail` $= 1$) using only information known before the inspection begins.

Our team builds a clear and repeatable pipeline. The pipeline first validates the dataset schema and cleans both the training and testing data in the same way. It then engineers new features while keeping the time constraint intact. After that, it benchmarks several models, starting from simple interpretable baselines and moving to more complex tree-based ensembles.

We train and tune all models using stratified cross-validation within the training set. The test set remains untouched until the final evaluation to measure how well the model generalizes. All steps run in a single reproducible script. Each step is logged to ensure transparency and auditability.

2.2. Data Cleaning

We apply a consistent cleaning process to both the training and testing datasets. This prevents bias between the two splits. Our code first checks that all required columns exist and removes any duplicate `inspection_id`.

We standardize all text fields. Each string is trimmed, converted to lowercase or title case, and mapped into consistent categories. The `results` field is reduced to a small, clear set of outcomes such as "pass," "fail," and "out of business."

City names are normalized, and a new flag marks whether a facility is located in Chicago. ZIP codes are converted to five-digit strings, and invalid or missing ones are set to NA.

We parse the `inspection_date` column into a proper date format. From it, we extract the inspection year, month, and weekday.

Risk labels are encoded as ordered integers, where high risk receives the largest value. Violation notes are parsed from semi-structured text to count the number of cited violations. When the field contains non-informative placeholder strings, the parser applies a conservative rule and assigns a count of 1. When the field is truly missing (NA), the count is 0. We keep only this numeric count to avoid text leakage.

Finally, we check latitude and longitude values. Points outside the reasonable Chicago range (41.60–42.10° N, 87.95–87.50° W) are marked as missing.

After cleaning, each dataset contains 18 columns. Restaurants remain the most common facility type. The data span 2010–2025 and cover all three risk levels and 19 inspection types.

2.3. Modeling Plan and Justification

We use a small but diverse set of models that work well with mixed tabular data. Each model serves a clear purpose in our benchmarking process.

1. **Logistic Regression**: This model offers a simple and interpretable starting point. It produces calibrated probabilities and helps us measure how much more complex models improve performance. We encode categorical features and apply ℓ_2 regularization to prevent overfitting. The model estimates the probability of failure as:

$$\Pr(y=1 \mid x) = \sigma\left(\beta_0 + \sum_j \beta_j x_j\right),$$

   where σ is the logistic function.

2. **Decision Tree**: This model captures simple feature interactions and threshold effects. It produces decision rules that are easy to interpret.

3. **Random Forest**: This model combines many decision trees to reduce variance. It is more stable than a single tree and handles outliers and correlated features effectively.

4. **Gradient-Boosted Trees**: These models often achieve the best performance on structured data. They build trees sequentially, learning from previous errors, and can model complex nonlinear relationships with built-in regularization.

We divide our features into three main types. Continuous variables include latitude, longitude, `viol_count`, and calendar fields such as year and month. The ordinal variable is `risk_ord`, which represents the inspection risk level. Categorical variables include `facility_type`, `inspection_type`, ZIP code, and `city_is_chicago`. We use one-hot encoding and handle unseen categories safely.

We do not use the raw `violations` text to avoid information leakage. Future versions of the model may use text features with careful cross-validation.

2.4. Training, Tuning, and Evaluation

We keep the provided test set completely separate from model development. The test data are used only once, at the very end, to measure real-world performance. All training, tuning, and model selection happen inside the training set.

We train each model using stratified k-fold cross-validation with k=5. This method keeps the same fail/pass ratio in every fold. It helps us make better use of limited data and produces stable performance estimates.

We tune model hyperparameters with either randomized search or grid search. The search covers key parameters such as tree depth, learning rate, minimum child weight, and subsampling rate. These ranges are chosen based on prior experience with tabular datasets.

Inspection failures are less common but more important to predict. Because of this imbalance, we report both threshold-free and threshold-based metrics. For threshold-free evaluation, we use ROC–AUC and PR–AUC to measure ranking quality. For threshold-based evaluation, we report Recall, Precision, and F_1 for the fail class. We also check model calibration with reliability curves. These plots show how well predicted probabilities match observed outcomes. To support interpretability, we will visualize feature importances and SHAP values.

We address class imbalance in two main ways. First, we apply class-weighted loss functions, such as `class_weight='balanced'` in logistic models or `scale_pos_weight` in gradient boosting. Second, we adjust classification thresholds on validation folds to balance recall and precision. We avoid oversampling within folds unless it becomes strictly necessary. This prevents data leakage and keeps evaluation honest.

2.5. Validation Plan

We select models and thresholds by stratified 5-fold CV on the training set. We then run a single, final evaluation on the held-out test set. A gap between CV and test indicates overfitting or drift; in that case we will tighten regularization, simplify features, or adopt temporal CV.

2.6. Implementation and Reproducibility

We implement the full pipeline as a single script with fixed seeds and logged configs. We version the cleaning function and apply it identically to train/test. We export cleaned CSVs, a meta-schema JSON, and EDA plots. Unknown categorical levels are routed to an explicit "unknown" bucket.

References

City of Chicago. Food Inspections: City of Chicago Data Portal. https://data.cityofchicago.org/Health-Human-Services/Food-Inspections/4ijn-s7e5, 2025. Accessed September 26, 2025.