Why You Must Handle Duplicates Before Outliers, Scaling, or Modeling

When your dataset contains a **significant portion of duplicates** — like **96,187 out of 938,583 rows (~10%)** — it is critical to resolve them *before* any other preprocessing steps such as:

- **Outlier detection**
- **Feature scaling**
- **Train–test splitting**
- **Encoding categorical features**

> Ignoring duplicates early on can seriously skew your downstream pipeline.

What Happens If You Don't Deduplicate First?

Effect	Explanation
Outlier methods get biased	Techniques like IQR or modified Z-score will miscalculate spread by treating duplicates as valid, reinforcing values that might otherwise be seen as outliers.
Scalers get skewed	Min–Max, Z-Score, and others will calculate incorrect ranges and distributions due to repeated values.
Leakage between train/test	If duplicates are split across training and testing, your model may appear to perform better than it should.
False sense of accuracy	Your classifier may memorize repeated records instead of generalizing, leading to overfitting and inflated performance metrics.

> **Bottom line:** Always handle duplicates first — it’s fast, easy, and protects the integrity of every step that follows.

Summary of Data Deduplication Techniques to Improve Model Accuracy

This guide outlines six key techniques to handle duplicates in your dataset, detailing what each method does, why it's worth trying, and how it can contribute to better model performance and reliability.

---

1. Direct Removal of Exact Duplicates

What it is:
Remove rows that are completely identical using `pandas.drop_duplicates()`.

Why try it:

- Eliminates over-represented rows that can distort training.
- Prevents data leakage into test sets.

How it helps:
Reduces overfitting and creates a fairer, more generalizable dataset.

---

2. Train–Test Aware Deduplication (or Grouped Cross-Validation)

What it is:
Assign entire *duplicate groups* (i.e. identical rows) to only **one side** of the split — either the training set **or** the test set — never both. For cross-validation, use grouping logic like `GroupKFold` to ensure the same rule applies across folds.

Why use it:

- Prevents **leakage**: no row (or its identical copy) ends up in both train and test.
- Preserves **natural frequency** for duplicates that reflect real-world distributions.
- Maintains **evaluation honesty**, especially when duplicates are frequent.

How it helps:
Avoids "cheating" by ensuring the model never sees test data during training — even indirectly via repetition. This leads to more **realistic generalization** performance and avoids overly optimistic accuracy.

---

3. Clustering-based Deduplication

What it is:
Group similar rows based on their feature values using clustering algorithms (e.g., DBSCAN), and remove near-duplicates.

Why try it:

- Effective for large, structured datasets with **numerical and categorical features**.
- Automatically detects clusters of similar data, removing redundant records.

How it helps:
Reduces noise, eliminates near-duplicates, and improves generalization by ensuring that the model doesn't overfit on repeated data.

---

4. Instance Weighting

What it is:
Assign lower weights to duplicate records during model training to reduce their impact while keeping all data.

Why try it:

- Preserves full data distribution.
- Useful when repetitions may be informative but shouldn't dominate.

How it helps:
Balances learning from all data while reducing overrepresentation effects.

---

5. Combined Deduplication: Exact Removal + Clustering

What it is:
First remove exact duplicates, then apply clustering (e.g., DBSCAN) to identify and eliminate near-duplicates in feature space.

Why try it:

- Double-layer defense: fast exact removal + intelligent group-level filtering.
- Combines the strengths of both approaches.

How it helps:
Maximizes deduplication effectiveness — catching both low-hanging duplicates and nuanced repetition patterns that could bias your model.
