**1. Introduction**

This study uses a machine predictive‑maintenance dataset (10,000 records, 9 sensors/features + identifiers, highly imbalanced target) to demonstrate how data leakage inflates model performance. We preprocess the data, train models with/without leakage, evaluate their metrics, then research leakage‑detection techniques.

**2. Data Preprocessing**

**a) Load the Data**

* **Code**: pd.read\_csv('predictive\_maintenance.csv') loads 10 000 rows × 10 columns .
* **Initial inspection** via df.info() confirms no missing values and data types (3 float, 4 int, 3 object) .

**b) Explore for Leakage**

1. **Target imbalance**: 9 661 non‑failures vs. 339 failures (≈3.4% failure) .
2. **Failure Type** feature uniquely maps to Target (correlation 0.959; “No Failure” → 0, all other types → 1) .
3. **Machine Type** (“H”, “L”, “M”) shows near‑zero correlation (–0.005) with Target .
4. **Numerical sensors** have low correlations (<0.20) with Target .

**Leakage source**: the “Failure Type” column directly encodes whether a failure occurred.

**c) Address Leakage**

* **Dropped columns**:
  + Identifiers: UDI, Product ID
  + Leaky: Failure Type, its binary mapping, plus highly correlated

Process temperature [K], Rotational speed [rpm] .

* **Encoding**: one‑hot on Type (drop first) after train/test split to avoid leakage .
* **Scaling**: StandardScaler on numerical features post‑split

**d) Document Findings**

| **Step** | **Action** |
| --- | --- |
| Missing values | None found; no imputation needed |
| Removed leaky features | “Failure Type”, its binary map |
| Removed identifiers | UDI, Product ID |
| Removed correlated cols | Process temperature [K], Rotational speed [rpm] |
| Encoded categorical | One‑hot on Type |
| Scaled numerical | StandardScaler on 3 numeric sensors |

**3. Model Training with and Without Leakage**

**a) Train/Test Split (80 / 20 Stratified)**

* **Purpose of stratification**: Ensures that both training and test sets preserve the original class imbalance (≈3.4% failures)
* **Resulting sets** :

**Training**: 8 000 samples (≈274 failures, 7 726 non‑failures)

**Test**: 2 000 samples (≈65 failures, 1 935 non‑failures)

**b) Models**

| **Model Type** | **Features Used** | **Purpose** |
| --- | --- | --- |
| **Logistic Regression** | **Clean features only** | **Baseline linear classifier without leakage** |
| **Gaussian Naive Bayes** | **Clean features only** | **Baseline generative classifier without leakage** |
| **Logistic Regression** | **Clean + “Failure Type”** | **Demonstrate effect of direct leakage** |
| **Gaussian Naive Bayes** | **Clean + “Failure Type”** | **Demonstrate effect of direct leakage** |

**4. Model Evaluation**

| **Model** | **Leakage?** | **Accuracy** | **Precision** | **Recall** | **F1‑score** |
| --- | --- | --- | --- | --- | --- |
| **Logistic Regression** | **No** | **0.9665** | **0.6667** | **0.0294** | **0.0563** |
| **Naive Bayes** | **No** | **0.9660** | **0.5000** | **0.0441** | **0.0811** |
| **Logistic Regression + SMOTE** | **No** | **0.7340** | **0.0930** | **0.7794** | **0.1661** |
| **Naive Bayes + SMOTE** | **No** | **0.7625** | **0.1018** | **0.7647** | **0.1796** |
| **Logistic Regression** | **Yes** | **0.9990** | **1.0000** | **0.9706** | **0.9851** |
| **Naive Bayes** | **Yes** | **0.9990** | **1.0000** | **0.9706** | **0.9851** |
| **Logistic Regression + SMOTE** | **Yes** | **0.9985** | **0.9851** | **0.9706** | **0.9778** |
| **Naive Bayes + SMOTE** | **Yes** | **0.9990** | **1.0000** | **0.9706** | **0.9851** |

**5. Analysis of Results**

* **Clean models (Logistic Regression, Naive Bayes):** achieve high overall accuracy (~96.6%) but fail to detect most failures (recall 3–4%). Accuracy 0.966, but very low recall (0.03–0.04) and F1-score (0.06–0.08) for failures (`Target` = 1).Reflects the imbalanced dataset (9661:339) and absence of leakage, making failure prediction challenging.
* **Leaky models (with “Failure Type”) :** Near-perfect metrics (accuracy ~0.999, recall ~0.971, F1 ~0.985). `Failure Type` directly reveals `Target` (correlation: 0.959389), causing unrealistic performance.
* **SMOTE (Clean):** Lower accuracy (0.734–0.76) but higher recall (0.765–0.779) and F1 (~0.166–0.180), improving failure detection.
* **SMOTE (leakage):** Near-perfect metrics (accuracy ~0.999, recall ~0.971, F1 ~0.98).Failure Type` directly reveals `Target` (correlation: 0.959389), causing unrealistic performance.

**6. Data Leakage Detection Techniques**

| **Technique** | **Description** | **Limitations** |
| --- | --- | --- |
| **1. Correlation & Crosstab** | **Compute feature–target correlations; near‑perfect mappings flag leakage.** | **May miss non‑linear leakage; Requires understanding whether high correlations are legitimate (e.g., `Torque` as a predictor) or leakage (e.g., `Failure Type`).** |
| **2. Random Forest Feature Importance** | **Train RF and inspect high‑importance features; unexpected top features hint at leakage.** | **Legitimate predictors can also rank high; overfitting may distort importances on small data.** |
| **3. Model Performance Discrepancy** | **1. Train two versions of the model: one with all features, one dropping suspected features. 2. Evaluate on held‑out test data. 3. A dramatic performance jump when including a feature suggests leakage.** | **- Requires retraining: Needs extra experiments. - Confounding factors: Performance gains could stem from legitimate predictive power, not necessarily leakage.** |

**Implementation Notes:**

* In our code, we computed both Pearson correlations and categorical crosstabs; “Failure Type” showed a perfect separation (100% of failure cases belonged to non-‑“No Failure” categories).
* We trained a Random Forest (100 trees) and saw “Failure Type” importance ≈ 0.78 vs. next feature ≈ 0.04.
* Finally, including vs. excluding “Failure Type” in logistic regression yielded a jump in test F1 from ~0.06 to ~0.98, confirming leakage.

**7. Conclusion**

**Data leakage fundamentally undermines the validity of machine‑learning experiments by allowing models to “cheat” — learning from information that would not be available at prediction time. In this assignment:**

1. **Identification of Leakage**
   * **We discovered that the Failure Type feature was essentially a proxy for the target (failure), because every non‑zero failure type corresponded directly to a failure event.**
   * **Simple correlation analysis (r ≈ 0.96) and cross‑tabulation immediately flagged this as a leakage risk.**
2. **Impact on Model Metrics**
   * **Without leakage, our logistic regression and Naive Bayes models achieved ~96.6% accuracy but dismally low recall (<5%) on the minority “failure” class—revealing their inability to generalize to rare events.**
   * **With leakage, the same algorithms reported ~99.9% accuracy and ~0.98 F1-score, a dramatic but spurious improvement driven entirely by the leaked feature.**
   * **SMOTE balancing improved recall at the cost of overall accuracy, highlighting the trade‑off when addressing class imbalance properly.**
3. **Practical Lessons**
   * **Always perform thorough EDA: Compute feature–target correlations, examine distributions, and cross‑tabulate categorical values before selecting inputs.**
   * **Use domain knowledge: Recognize when a feature (like Failure Type) encodes future or post‑event information that won’t be available at deployment.**
   * **Pipeline discipline: Apply encoding, scaling, and feature removal inside cross‑validation or post‑split pipelines to avoid inadvertent information bleed.**