DPF Soot Analysis

Assignment for initial round of internship opportunity at Tensor Planet

S Sanjeev Srinivas

1. Data Generation Approach

I approached this task by first analysing the physics of soot accumulation and the risks of engine derate events in commercial diesel vehicles. My main goal for the Data Generation phase was to build a dataset that felt structurally consistent but not "too algorithmic."

I imposed stochastic randomness and injected specific data quality errors (such as stuck sensors and null values) to simulate real-world telemetry imperfections. Also, instead of independent random variables; I modelled dependencies to reflect actual driver behaviour. A key example is idle time: instead of being fully random, it is directly dependent on trip distance with specific time thresholds. This ensures that "long-haul" trips include realistic driver rest periods, preventing the model from learning artificial patterns.

Beyond simple behavioural patterns, I focused on maintaining thermodynamic consistency across the sensor array. In a real diesel engine, variables like RPM, Exhaust Flow, and Differential Pressure are physically coupled, not independent. To reflect this, I implemented a dependency chain where Engine Load and RPM drive the Exhaust Flow Rate, which in turn acts as a multiplier for the Differential Pressure reading.

A key implementation detail was the simulation of exothermic regeneration events. Instead of treating regeneration as a simple Boolean switch, I mode led the temperature spike where the DPF Inlet Temperature exceeds the DOC Inlet Temperature due to fuel oxidation. This nuanced feature allows the machine learning model to learn the physical signature of a regeneration event (a negative temperature delta) rather than relying solely on explicit maintenance flags, which are often delayed or missing in real-world telemetry. Finally, to address the requirement for System Robustness, I introduced "Stuck Sensor" logic, where specific sensors (like Differential Pressure) would freeze and repeat their last known value for extended periods, simulating electrical faults.

2. Data Engineering Decisions

Moving from raw telemetry to a model-ready dataset required aggressive feature engineering to expose the underlying signal hidden in the noise I generated. I built a pipeline

that generated 44 advanced features, focusing on capturing temporal dynamics rather than static snapshots.

- Temporal Features: I engineered rolling statistics (mean and standard deviation over 10-minute windows) for Exhaust Temperature and RPM. This is critical because a single high-temperature reading might be noise, but a sustained high-temperature trend indicates a heavy load or regeneration event.
- Interaction Features: I created physics-based interaction terms, such as Speed × Load and Temp × Pressure. These features act as proxies for the total energy output of the engine, which correlates strongly with soot production rates.
- Handling Imbalance: The raw dataset reflected reality: actual failures were rare (approx. 1.4%). Training a model on this directly would lead to a bias towards the majority class (healthy trucks). To counter this, I implemented SMOTE (Synthetic Minority Over-sampling Technique) to synthetically balance the training data. This was a crucial engineering decision that allowed the model to learn the subtle patterns of failure without being overwhelmed by the noise of healthy operations.

3. Problem Framing and Modelling Rationale

For the modelling phase, I framed this as a Classification problem where the goal was to predict a "High Risk" state (soot load > 4.5 g/L) based on driving telemetry. This framing moves away from trying to regress an exact soot value—which is noisy and stochastic—and focuses instead on providing actionable "Red/Green" alerts to fleet managers.

I evaluated three gradient boosting algorithms for this task: XGBoost, LightGBM, and CatBoost. After initial training and hyperparameter tuning (GridSearch over 729 combinations), the results presented a clear trade-off:

- CatBoost: Demonstrated superior Precision (21.3%). This model was excellent at minimizing false alarms, meaning if it flagged a truck, that truck was more in trouble.
- XGBoost: Achieved the highest Recall (23.4%). This model was more aggressive, catching a significantly larger number of actual failures, though at the cost of flagging some healthy trucks as risky.

The trade-off decision: Leaning into recall in many data science contexts, high precision is the gold standard. However, applying engineering judgment to the specific economics of fleet operations flipped this priority. I leaned towards prioritizing Recall (minimizing False Negatives) for one critical reason: the asymmetry of cost.

- The cost of a False Positive: Triggering a "maintenance check" alert when a truck is actually fine is relatively cheap. Most modern fleets have routine inspection intervals, and a quick diagnostic plug-in takes 10–15 minutes of mechanic time.

- The Cost of a False Negative: Missing a critical soot event is catastrophic. If a truck enters "derate mode" (limp mode) on a highway or blows a DPF, the costs skyrocket—towing fees, emergency repairs, and crucially, missed delivery penalties and downtime.

Given this reality, I selected XGBoost as the production model. While CatBoost offered a "cleaner" alert stream, XGBoost provided the necessary early warning system safety net. It allows operators to catch 175% more potential failures than the baseline. I accepted a higher false alarm rate as the "cost of doing business" to ensure we don't lose engines. This approach shifts the strategy from "perfect accuracy" to "risk mitigation," ensuring proactive interventions happen before the damage is irreversible.

4. Production and Robustness Considerations

A model is useless if it cannot be deployed reliably. I designed a production architecture using FastAPI for high-performance serving, wrapped in a Docker container to ensure environment consistency.

- Latency & Caching: To handle the high volume of telemetry data, I integrated a Redis cache layer. Since soot accumulation is a slow process, we don't need to re-compute predictions for the same vehicle every second. Caching predictions with a 5-minute TTL (Time-To-Live) reduced API latency from ~30ms to ~5ms for repeated requests, a massive efficiency gain for fleet-wide monitoring.
- Monitoring: I implemented Prometheus metrics to track prediction distributions in real-time. This allows us to detect "data drift"—if the model suddenly starts flagging 50% of the fleet as "High Risk," we know there's likely a sensor calibration issue or a bad batch of fuel, rather than a sudden fleet-wide DPF failure.
- Experiment Tracking: I utilized MLflow to log every hyperparameter and model version. This ensures reproducibility; if a new model version performs poorly in production, we can instantly roll back to a specific, previously validated artifact.

5. Business Impact Analysis

The technical metrics translate directly into operational savings. Based on the validation set, the final XGBoost model detects 359 out of 1,591 high-risk failures, compared to just 136 detections by the baseline heuristic.

- Breakdowns Prevented: This delta represents 223 additional vehicle breakdowns prevented. If we estimate the cost of an on-road failure at roughly $2,000 (towing + urgent repairs + delay penalties), this model saves the fleet approximately $446,000 in avoided costs over the evaluation period alone.
- Operational Efficiency: Furthermore, the feature engineering reduced false alarms from 3,722 (baseline) to 1,162. This is a 69% reduction in unnecessary mechanic trips, significantly reducing maintenance labor overhead and allowing staff to focus on genuine issues.

6. Recommendation and Justification

The deployed XGBoost model , on further improvement, specifically configured with the optimized decision threshold of 0.1249 derived from the precision-recall analysis, can prove to be a very useful tool to tackle the issue of Soot accumulation.

While a 20.3% precision might seem low on paper, it is functionally optimal for this domain. It effectively acts as a "1-in-5" rule: for every five alerts the system generates, one will be a critical save that prevents a major breakdown. For fleet operators, checking four healthy trucks to save one engine is a highly profitable trade-off.

For future direction, I would integrate the "Stuck Sensor" logic directly into the inference pipeline as a pre-check rule. Currently, the model handles noise well, but explicitly filtering out frozen data streams before they reach the model would further reduce false positives and improve trust in the system. And make the production software and UI better to be more user-friendly.