Predictive Maintenance for Vehicles: A Machine Learning Approach

Féeryna Fihento, Juliette Ibrahim, Sandro El Khoury

ESILV A4 MMN

Contents

1 Introduction

1.1 Project Context

In 2023, the average annual maintenance cost reached approximately 1,475 USD for 15,000 miles. Thus, unnecessary maintenance creates a financial burden, especially considering that over half of households can't afford an unexpected repair costing more than 1,000 USD. For context, routine maintenance, typically recommended every 5,000 to 7,500 miles, includes services like oil changes, tire rotations, and various inspections. However, these conventional approaches tend to overlook crucial factors such as driver behavior, environmental conditions, and the actual degradation or failure state of components. As a result, many vehicle owners either perform premature and unnecessary maintenance while also overlooking early warning signs, eventually facing major and costly failures. As a result, many vehicle owners perform unnecessary maintenance or ignore the main issue, leading to big failures. Thankfully, recent researches have explored the use of machine learning models to predict vehicle maintenance needs more accurately.

Our proposed solution aims to improve the prediction of vehicle maintenance needs by leveraging vehicle-specific characteristics. We therefore use predictive maintenance to solve this problem by using data that allows us to estimate when a vehicle will need maintenance. Thanks to machine learning, we analyze the vehicle's information and can detect the first signs of wear.

This project was based on a Kaggle dataset that describes the characteristics of vehicles to predict their maintenance needs. We used different learning models and compared them in order to evaluate their ability to provide accurate predictions and reduce unexpected failures. This approach will enhance the maintenance need prediction process in a way that reduces downtime and ultimately play a role in cost-effective vehicle management.

1.2 Objective and Motivation

The goal of this project is to build a model that predicts whether a vehicle is likely to need maintenance, formulated as a binary classification task. The aim is not only to achieve high accuracy, but also reliable predictions that can support real operational decisions, which requires handling class imbalance, choosing meaningful metrics and evaluating generalization on new data.

As future engineers, the two most interesting aspects that this project brings us are:

- operational efficiency: we can predict maintenance needs early so interventions can be proactively scheduled, reducing downtime and repair costs.

- anticipation of needs and resources: maintenance teams and spare parts can be planned more efficiently when risk estimates are available for each vehicle.

The objective of this project is not only to achieve high performance, but also to assess whether the data provides a sufficient predictive signal and whether the resulting model can realistically support system managers and operational planning. This project is directly part of our major modeling and numerical mechanics within which predictive analysis plays an essential role in the evaluation of the integrity of the structure note, the wear of the component and the reliability of the system. Indeed, maintenance is a crust factor in the aeronautical and automotive industries because it represents an essential safety imperative. Being able to predict

the mechanical degradation of a system thanks to data is part of engineering's challenges and this machine learning project, the basic principles of mechanical engineering with modern computing methods, a synergy at the heart of our training at ESILV and our professional aspirations in the automotive, aeronautical and aerospace sectors.

2 Dataset Description

2.1 Source and Characteristics

For this project, we used the dataset `vehicle_maintenance_data.csv`, from Kaggle. It consists of records describing a fleet of vehicles, including technical specifications, mileage indicators, and condition-related characteristics. Each observation corresponds to a single vehicle over a given period, and all samples are annotated with a binary variable indicating whether maintenance was required. This dataset contains 30,000 cases, which provides a sufficiently large sample to train machine learning models with stable statistical properties. Most features are numerical, with a small number of categorical variables representing vehicle type or configuration. These variables capture measurable aspects of vehicle usage that could influence maintenance requirements, such as accumulated mileage, engine capacity, or performance indicators. Data availability is historical rather than real-time, and no explicit temporal ordering is used in this project. Consequently, the modelling approach is based on static risk prediction, rather than sequential forecasting, this means that the model looks at a snapshot of data (vehicle age, mileage, engine size, accident history, etc.) and predicts the risk of maintenance at that point in time.

2.2 Main Variables

The dataset includes several input variables that represent technical aspects of vehicle operation. Examples of typical features include:

1. Mileage or cumulative distance
2. Engine size or displacement
3. Fuel consumption indicators

These variables have different scales and distributions, which requires preprocessing before model training. Numerical features may exhibit skewness, while some categorical variables reflect structural differences between vehicle segments. No engineered variables were originally provided in the dataset. Feature engineering is there- fore performed during the modelling phase to enhance predictive power. First, we set up the environment, load the data set and first look at its size and contents before starting preprocessing and training machine learning models. This step is also known as the Data collection step: We gather all relevant historical and operational data that could influence maintenance needs:

Shape of dataset: (30000, 20)

	Vehicle_Model	Mileage	Maintenance_History	Reported_Issues	Vehicle_Age	Fuel_Type	Transmission_Type	Engine_Size	Odometer_Reading	Last_Servi
0	Truck	53723	Average	3	4	Diesel	Automatic	800	120202	202
1	Car	66530	Good	5	10	Petrol	Automatic	1500	111629	202
2	Car	50950	Good	0	3	Petrol	Manual	2500	39116	202
3	Truck	34951	Poor	3	1	Petrol	Automatic	2500	20774	202
4	Motorcycle	35208	Poor	1	5	Diesel	Automatic	2000	124695	202

Figure 1: Preview of the dataset

```
Missing values per column:
 Vehicle_Model           0
Mileage                  0
Maintenance_History      0
Reported_Issues          0
Vehicle_Age              0
Fuel_Type                0
Transmission_Type        0
Engine_Size              0
Odometer_Reading         0
Last_Service_Date        0
Warranty_Expiry_Date     0
Owner_Type               0
Insurance_Premium        0
Service_History          0
Accident_History         0
Fuel_Efficiency          0
Tire_Condition           0
Brake_Condition          0
Battery_Status           0
Need_Maintenance         0
dtype: int64

Number of duplicate rows: 0
```

Figure 2: Missing values summary and duplicate rows.

The dataset seems clean because there are no missing values, no duplicates, and all numerical features fall within realistic ranges.

2.3 Target Distribution and Class Imbalance

Categorical columns are correctly identified, and the target variable is imbalanced with about 81% positive cases. This validates that the data is usable for modeling.

For reference, the target variable is a binary indicator identifying whether a vehicle required maintenance within the observed period. A value of 1 corresponds to vehicles that received maintenance, and 0 to vehicles that did not.

An inspection of the target distribution shows a significant class imbalance, as 0s are 4570 and 1s are 19430, with the majority class representing non-maintenance cases. This imbalance is problematic for classification, as most models tend to favor the majority class and underestimate the occurrence of maintenance events.

The observed imbalance justifies the use of specialized techniques to adjust class frequencies during training. In this project, oversampling using SMOTE (Synthetic Minority Oversampling Technique) was applied exclusively on the training data to prevent information leakage.

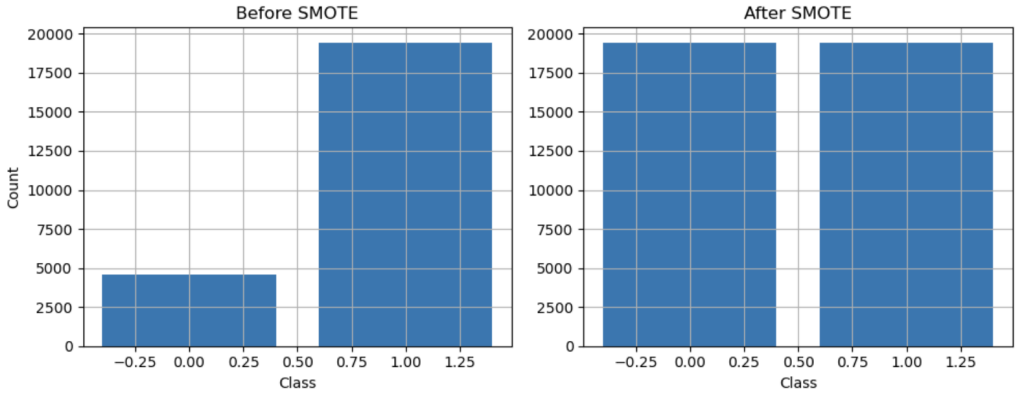


Figure 3: Class distribution of the target variable `Need_Maintenance`.

3 Data Preprocessing

3.1 Outlier Inspection

Outliers in numerical features were examined using boxplots. No outliers were detected.

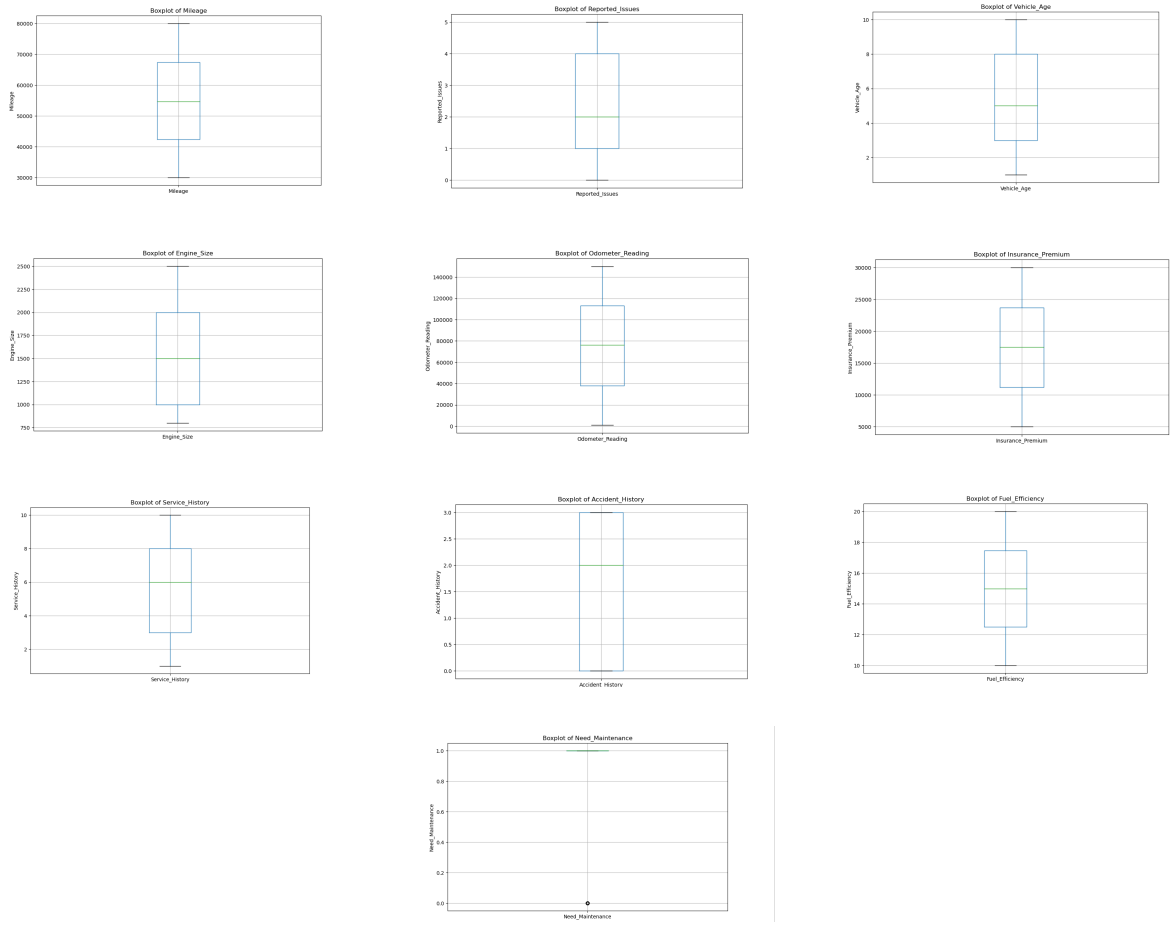


Figure 4: Boxplots to spot outliers or strange values

3.2 Normalization and Encoding

However, it is important to note that SMOTE is applied to numerical data, but the dataset has categorical variables like fuel type, transmission type, ... And to overcome this situation, we need to convert them into a numerical format. And in order to convert categorical data into numerical data, we perform encoding. One type of encoding is one-hot encoding, which is useful for converting nominal categorical variables and another is label encoding for converting ordinal categorical variables. Categorical variables were encoded using one-hot encoding, enabling their use in tree-based and linear models. Encoding was performed within the preprocessing pipeline to maintain reproducibility and to guarantee consistent transformation across cross-validation folds

To ensure consistent scaling of numerical features, standardization was applied during model training. Importantly, scaling was performed after the train/test split, and the scaler was fitted only on the training data to prevent information leakage.

.

3.3 Train/Test Split and Stratification

The dataset was partitioned into training and test sets using an 80/20 split, with stratification on the target variable to preserve the original class distribution in both subsets.

Stratification was essential given the imbalance observed in the target labels, as it prevents models from being trained and evaluated on distributions that differ from the overall dataset.

All subsequent preprocessing and model fitting steps were performed exclusively on the training partition, with the test set reserved for unbiased evaluation.

3.4 Imbalance Handling (SMOTE)

After the split, we applied balancing (SMOTE in our case) inside the training set. We chose to do this step after the training/testing to avoid data leakage, but also to preserve test set integrity (Because balancing is only for training, so the model learns better patterns.)

As observed previously, the dataset exhibited a strong class imbalance, with maintenance events representing a minority of observations. To mitigate the tendency of models to favor the majority class, the training data was balanced using SMOTE.

SMOTE generates new samples for the minority class until it balances out with the majority one.

Oversampling was carried out inside the cross-validation pipeline, ensuring that synthetic samples were generated only on training folds and never leaked into validation sets.

```
for name, clf in models.items():
    print("\n" + "=" * 70)
    print(f"Model: {name}")
    print("=" * 70)

    # Full pipeline: preprocessing + classifier
    pipe = ImbPipeline(steps=[
    ("preprocess", preprocess),          # impute + scale + one-hot
    ("smote", SMOTE(random_state=42)),   # balancing ONLY on train folds with SMOTE
    ("model", clf),                      # classifier
])
```

Figure 5: Code snippet showing SMOTE integration in the pipeline.

4 Feature Engineering

4.1 Engineered Features

The project used the original features available in the dataset, which describe different characteristics of each vehicle, such as mileage, age, fuel type, engine size and maintenance history. These variables were used directly as inputs for the machine learning models because they were already informative and did not require major modifications.

Categorical variables were encoded so that models could process them, and numerical variables were scaled after the train/test split to avoid information leakage. These steps ensured that all features were in the correct format and comparable in scale, which is important for models that use distance or gradient-based optimization.

Overall, the modelling process relied on the existing dataset structure, and the original features were preserved rather than replaced or transformed into new ones.

```python
# 1) Convert date columns to datetime
df["Last_Service_Date"] = pd.to_datetime(df["Last_Service_Date"], errors="coerce")         #conversion to date time
df["Warranty_Expiry_Date"] = pd.to_datetime(df["Warranty_Expiry_Date"], errors="coerce")

# 2) Create engineered numerical features from dates
#  We use a fixed reference date for reproducibility
reference_date = pd.to_datetime("2025-01-01")

df["Days_Since_Last_Service"] = (reference_date - df["Last_Service_Date"]).dt.days
df["Days_Until_Warranty_Expiry"] = (df["Warranty_Expiry_Date"] - reference_date).dt.days

# Clip negative values (if warranty already expired or service in the future)
df["Days_Since_Last_Service"] = df["Days_Since_Last_Service"].clip(lower=0)
df["Days_Until_Warranty_Expiry"] = df["Days_Until_Warranty_Expiry"].clip(lower=0)

# 3) Define feature columns (we keep engineered features, drop original dates + target)
FEATURE_COLUMNS = [
    col
    for col in df.columns
    if col not in ["Need_Maintenance", "Last_Service_Date", "Warranty_Expiry_Date"]
]

X = df[FEATURE_COLUMNS]
y = df["Need_Maintenance"]
```

Figure 6: Creation of `Days_Since_Last_Service` and `Days_Until_Warranty_Expiry`.

4.2 Purpose of feature engineering

In this project, feature engineering was limited because the dataset already contained clean and meaningful variables. The main goal was to make the data easier for the models to learn from, rather than creating complex new features.

Most original variables (such as mileage, age, or engine size) were kept as they were, since they already described important aspects of the vehicle. Some transformations, such as scaling and encoding, were applied to ensure that the models could use the data correctly.

Overall, feature engineering helped standardize the inputs and slightly improve model stability, but major feature creation was not necessary due to the quality and simplicity of the dataset.

```
print("Feature columns used:\n", FEATURE_COLUMNS, "\n")

# 4) Identify numerical vs categorical features
num_cols = X.select_dtypes(include=["int64", "float64"]).columns.tolist()
cat_cols = X.select_dtypes(include=["object"]).columns.tolist()

print("Numeric columns:\n", num_cols, "\n")
print("Categorical columns:\n", cat_cols, "\n")

# 5) Train/test split with stratification (important because of imbalance)
X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.2,
    stratify=y,
    random_state=42,
)

print("Training set size:", X_train.shape[0])
print("Test set size:", X_test.shape[0])



# 6) Preprocessing pipelines
# Numeric: impute missing with median + standardize
num_transformer = Pipeline(
    steps=[
        ("imputer", SimpleImputer(strategy="median")),
        ("scaler", StandardScaler()),
    ]
)

# Categorical: impute missing with most frequent + one-hot encode (dense for all models)
cat_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
    #("encoder", OneHotEncoder(handle_unknown="ignore", sparse_output=False))
    ("encoder", OneHotEncoder(handle_unknown="ignore", sparse=False))

])



preprocess = ColumnTransformer(
    transformers=[
        ("num", num_transformer, num_cols),
        ("cat", cat_transformer, cat_cols),
    ]
)
```

Figure 7: Preprocessing pipeline

5 Modelling Approach

We chose six base machine learning models for the evaluation and several advanced techniques were selected with the aim of improving performance.

5.1 Base Models

Here are the six selected models:

- **Logistic regression**: A linear model that uses a logistic function to model the probability of the binary result. It serves as a solid foundation.
- **Random Forest**: A defined method that constructs multiple decision trees during the training and displays the class mode (classification) of individual trees (robust and resistant to overfittin) .
- **Gradient Boosting**: This ensemble technique builds models in a sequantial way. Each model tries to corect the mistakes made priviously. Often reach hogh precision but may subject to overfitting if is not tuned corectly.
- **K-Nearest Neighbors (KNN)**: This non-parametric method classifies a data point as on the majority class among its k nearest neighbors in the space of the feature.

- **Decision tree**: A model like a tree model of decisions and their possible consequences. It is easy to interpret but can easily overfit the training data.
- **Gaussian Naive Bayes**: a probabilistic classifier based on the application of Bayes' theorem with strong assumptions of independence between characteristics.

5.2 Evaluation Metrics

We then evaluate the performance of the different models using three main metrics:

- **Accuracy**: proportion of correctly classified observations
- **F1-score**: harmonic mean of **precision** and **recall**, useful when classes are unbalanced
- **ROC-AUC**: area under the ROC curve, measuring the ability to rank positive cases higher than negative ones

For each model, these metrics were calculated on the test set. ROC-AUC was used as the main criterion for comparing models because this metric is less sensitive to class imbalance than accuracy. In addition, the F1-score is monitored to ensure that the minority class (necessary maintenance) was not ignored by the classifier.

```
Summary of metrics on test set:
LogisticRegression  ROC-AUC: 0.9898  Accuracy: 0.9607  F1: 0.9753
RandomForest        ROC-AUC: 1.0000  Accuracy: 1.0000  F1: 1.0000
GradientBoosting    ROC-AUC: 1.0000  Accuracy: 1.0000  F1: 1.0000
KNN                 ROC-AUC: 0.9165  Accuracy: 0.7797  F1: 0.8445
DecisionTree        ROC-AUC: 0.9998  Accuracy: 0.9997  F1: 0.9998
GaussianNB          ROC-AUC: 0.9895  Accuracy: 0.7480  F1: 0.8157
```

Figure 8: Performance table for the six base models (Accuracy, F1, ROC-AUC).

5.3 Advanced Techniques

Here are the advanced techniques that we use:

- **Principal Component Analysis (PCA) with Logistic Regression**: We applied PCA to reduce the dimensionality of the space of the feature before we train a Logistic Regression model. The goal was to evaluate if key predictive models could be captured in a lower dimensional space.
- **Ensemble methods**: Three types of ensemble models were tested:
  - **Voting Classifier**: Combines predictions from multiple models (logistic regression, Random Forest, Gradient Boosting) using a soft voting scheme (predicted mean probabilities).
  - **Bagging Classifier**: Uses bootstrap aggregation with Decision Trees as the basis estimator to reduce the variance.

  - **Stacking Classifier**: Using of a a meta-classifier (Gradient Boosting) in order to combine the predictions of basis estimators (Logistic Regression, Random Forest).

- **Hyperparameter tuning**: The most efficient model, Gradient Boosting, was subjected to a grid search ("GridSearchCV") to find the optimal combination of hyperparameters (number estimators, learning rate, maximum depth, subsample) to further improve its performance.

All models were trained within a pipeline that included the pre-processing steps (imputation and encoding) and the application of SMOTE oversampling on the training folds during cross-validation to mitigate the effects of class imbalance.

Model	ROC-AUC	Accuracy	F1-score
LogisticRegression	0.9898	0.9607	0.9753
RandomForest	1.0000	1.0000	1.0000
GradientBoosting	1.0000	1.0000	1.0000
KNN	0.9165	0.7797	0.8445
DecisionTree	0.9998	0.9997	0.9998
GaussianNB	0.9895	0.7480	0.8157
Voting	1.0000	1.0000	1.0000
Bagging	0.9998	0.9998	0.9999
Stacking	1.0000	1.0000	1.0000
PCA + LogisticReg	0.9896	0.9598	0.9746
Tuned GradientBoost	1.0000	1.0000	1.0000

Figure 9: Performance of advanced models: PCA+LR, Voting, Bagging, Stacking, and Tuned Gradient Boosting.

```
Tuned GradientBoosting — test set performance:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1143
           1       1.00      1.00      1.00      4857

    accuracy                           1.00      6000
   macro avg       1.00      1.00      1.00      6000
weighted avg       1.00      1.00      1.00      6000

Test ROC-AUC : 1.0
Test Accuracy: 1.0
Test F1-score: 1.0
```

Figure 10: Output of GridSearchCV for Gradient Boosting: best parameters and CV score.

6 Addressing Obstacles

During the project we ran into a few issues that slowed things down, and most of the fixes happened gradually as we figured out what was wrong. Nothing was solved in one step, we kind of tried things, saw results, and adjusted.

6.1 Data Imbalance

One big problem was that most rows were from the same class, roughly 80% or a bit more. If we didn't change anything, the model could simply predict the majority label and still look "accurate", which is useless for maintenance prediction.

We tried two things mainly: giving more weight to the minority class and applying SMOTE so that rare cases would appear more often during training.

6.2 Overfitting and Underfitting

We were also worried about the model learning the training data too well. Tree models tend to do that, especially when data is clean. Underfitting (the opposite problem) didn't really seem to happen here.

To check this, we used stratified cross-validation and later compared train and test performance. Scores were basically the same, so nothing weird showed up. We did some tuning on Gradient Boosting and Logistic Regression just to be safe.

```
Train vs Test performance:
Train accuracy: 1.0000, Test accuracy: 1.0000
Train F1:       1.0000, Test F1:       1.0000
Train ROC-AUC:  1.0000, Test ROC-AUC: 1.0000
```

Figure 11: Train vs. Test performance for Tuned Gradient Boosting (F1-score).

6.3 Parameter Optimization

For Gradient Boosting we tried different combinations of parameters such as number of trees, depth, learning rate and subsampling.

- `model__n_estimators`: [100, 200]
- `model__learning_rate`: [0.05, 0.1]
- `model__max_depth`: [3, 4]
- `model__subsample`: [0.8, 1.0]

We evaluated them with 3-fold cross-validation using F1-score. One setup got a perfect score, which looked nice, but honestly we didn't expect results that high.

- `model__learning_rate`: 0.05
- `model__max_depth`: 3

- `model__n_estimators`: 100
- `model__subsample`: 0.8
- Best CV F1-score: 1.0000

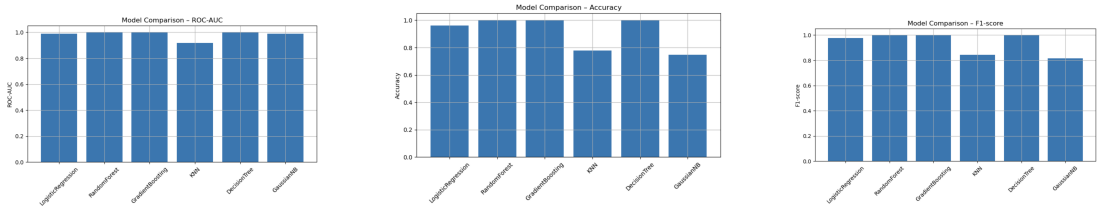
7 Results and Analysis

7.1 Model Performance Comparison

We compared models with ROC-AUC, accuracy and F1-score.

Tree-based models (Random Forest, Gradient Boosting, Decision Tree) were clearly the best ones. Ensemble methods were also pretty good.

KNN and Naive Bayes didn't perform well, probably because of high dimensionality after encoding and the assumptions behind these models.



(a) ROC-AUC (b) Accuracy (c) F1-score

Figure 12: Bar charts comparing all models across the three evaluation metrics.

7.2 Best Model Selection

We picked Gradient Boosting as the final one because the scores were stable and it reacted well to SMOTE. Since it learns step-by-step and tries to fix past errors, it behaves better than a single tree. Random Forest also worked well but doesn't correct errors between trees.

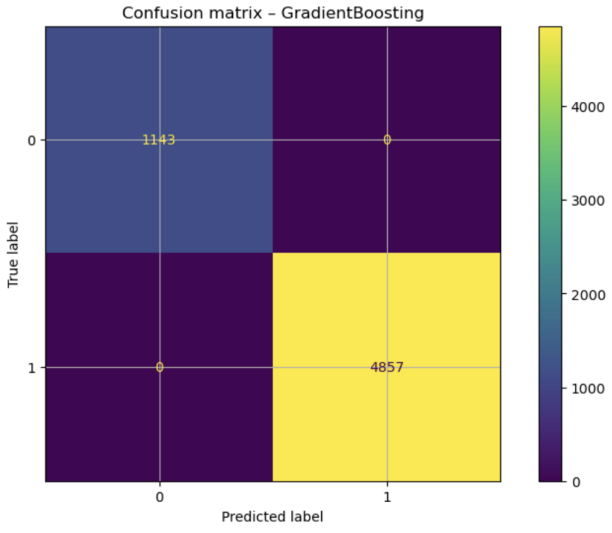


Figure 13: Confusion matrix of the final Tuned Gradient Boosting model.

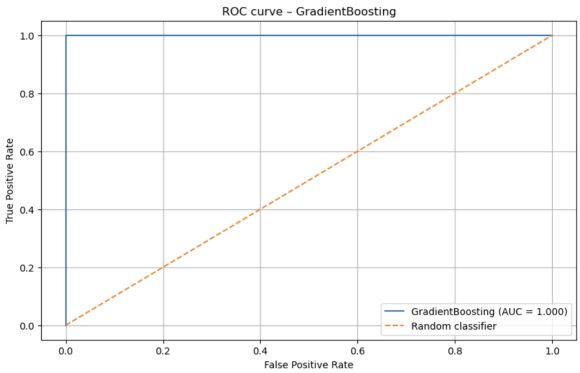


Figure 14: ROC curve of the final Tuned Gradient Boosting model (AUC = 1.0).

8 Feature Importance

8.1 Most influential features

We checked which features influenced predictions the most. The top ones were Reported Issues and several battery-related variables, plus brake condition. Battery features showing up a lot probably means battery health is a strong indicator of vehicle condition.

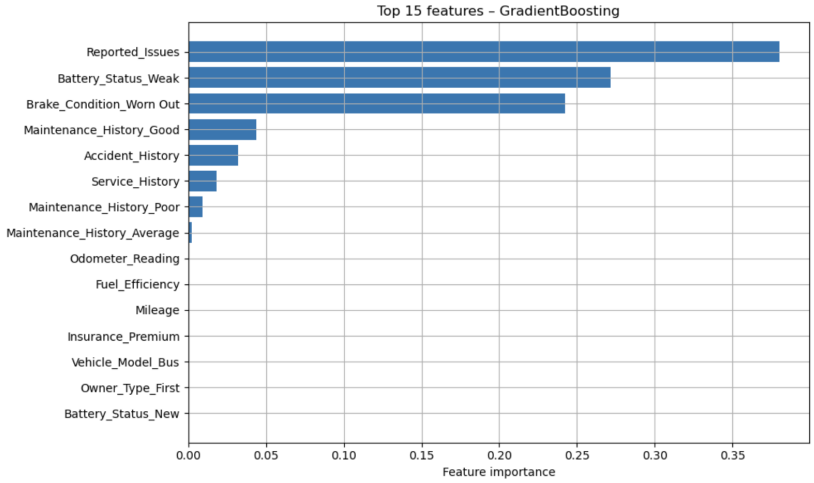


Figure 15: Feature importance bar plot from the final Gradient Boosting model.

8.2 Interpretation

Reported Issues makes sense as the most important feature because past failures usually predict future ones. Brake and battery status also mattered, which highlights the value of inspection data instead of only usage stats.

9 Business Implications

9.1 Value for Predictive Maintenance

The model can warn us before a failure happens. That means scheduling work earlier, reducing downtime and avoiding situations where everything stops unexpectedly.

9.2 Operational Impact

Practically, this helps deciding which vehicles need attention first, organizing work, keeping enough spare parts, and avoiding emergency repairs. The model doesn't replace human judgment but it gives useful hints.

10 Conclusion

10.1 Summary of Findings

This project explored the role of ML in predicting vehicle maintenance needs. While traditional methods rely on a fixed schedule or mileage coverage that can be insufficient, machine learning models can propose an approach to predict maintenance based on different characteristics of vehicles such as vehicle details (Model, Age, Fuel Type, Transmission, Engine Size, Owner Type), but also usage and history (Mileage, Odometer Reading, Service and Maintenance History, Accident History, Reported Issues), and condition and performance (Fuel Efficiency, Tire, Brake, Battery status), with other information (Last Service Date, Warranty Expiry, Insurance Premium, Need Maintenance flag). However, the dataset we chose was highly imbalanced, so we made sure to fix this problem with SMOTE. The modelling pipeline included preprocessing steps, scaling applied after the train–test split to avoid information leakage, and class balancing applied only on the training data. These choices ensured that the models were trained in a controlled environment.

We also tested several algorithms to maximize the accuracy of the results: Logistic regression, Random Forest, Gradient Boosting, K-Nearest Neighbors (KNN), Decision Tree and GaussianNB.

Those ML models were trained and compared using accuracy, F1-score and ROC-AUC. Results showed a clear hierarchy between approaches: While Models such as KNN and Gaussian Naive Bayes performed not so bad,tree-based models, and especially Gradient Boosting, provided the most reliable predictions. The model consistently achieved high scores on both training and test sets, indicating that the dataset contains strong and easily separable patterns.

The most relevant features were related to previously reported issues and component-level indicators such as battery and brake condition. This highlights that past events and inspection results carry more predictive value than general vehicle attributes (which is logic with the correlation map showing that each parameter is independant from each other).

These results confirm that a data-driven approach can help anticipate maintenance needs rather than reacting once failures occur. In practice, such a system could support the planning of inspections, resource allocation and prioritisation of high-risk vehicles, potentially reducing downtime and unplanned interventions.

To sum up everything that has been stated, the results can help in enhancing maintenance strategies for both vehicle owners and fleet managers to minimise downtime and make maintenance cost-effective.

10.2 Limitations

However, the very high performance observed in this project should be interpreted with caution. Real environments contain noise, missing information and evolving system behaviour, which may reduce model performance over time. A deployed system would therefore need regular updates, monitoring and retraining to remain reliable.

Overall, the project demonstrates that predictive maintenance for vehicle fleets is technically feasible when appropriate data is available. The approach provides meaningful insights, identifies key factors influencing maintenance needs, and shows that machine learning can contribute to more efficient and proactive maintenance strategies.

Even though results were very good, they might not generalize to messy data in real situations. Real systems change over time, so a deployed model would need to be updated to stay useful.

References

1. Kaggle. (2024). *Vehicle Maintenance Data*. `https://www.kaggle.com/datasets/chavindudulaj/vehicle-maintenance-data`