IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 11, November 2026

AI Public Infrastructure Failure Predictor Using Machine Learning

Mr. Kunal Bhatu Kuwar[1], Prof. Manoj V. Nikum[2]

Research Scholar MCA, Shri Jaykumar Rawal Institute of Technology, Dondaicha, India[1]

HOD MCA & Assistant Professor, Shri Jaykumar Rawal Institute of Technology, Dondaicha, India[2]

Abstract: Public infrastructure such as roads, bridges, water pipelines, drainage lines and electrical grid nodes is usually maintained after something has already gone wrong. This paper describes the design and a prototype evaluation of the Public Infrastructure Failure Predictor, a system that combines inspection records, maintenance history, sensor readings, asset age, usage load and weather exposure to estimate how likely each asset is to fail within the next twelve months. Machine learning models (logistic regression, Random Forest and gradient boosting, with XGBoost and LSTM planned for the full system) produce a risk score for every asset, and the main contributing factors are reported alongside it. A Spring Boot backend with JWT-based access control and a PostgreSQL/PostGIS database feeds a React dashboard that shows risk-ranked assets on an interactive map. Because real municipal failure records were not available, the models were tested on a synthetic dataset of 6,000 assets. On held-out data the best model reached a ROC-AUC of 0.83, and ranking assets by predicted risk located 46% of the failures within the top 10% of inspected assets, against about 10% for random inspection. These figures show that the pipeline works end to end, but they must be confirmed on real data before any operational claim is made.

Keywords: Predictive maintenance; Infrastructure failure prediction; Machine learning; Risk scoring; Geospatial dashboard; Smart city

I. INTRODUCTION

Most people only think about public infrastructure when it stops working. A water main bursts and a street floods, a bridge bearing is found cracked during a routine walk-over, a transformer fails in the evening peak and a whole neighbourhood loses power. Municipal bodies in India look after thousands of such assets with small maintenance teams and budgets that are fixed well before the year begins, so every failure that was not planned for takes money and staff away from something else.

In practice, maintenance is still either reactive or calendar-driven [1]. Reactive maintenance means the asset is repaired after it fails, which usually involves emergency crews, traffic or supply disruption and, in the worst cases, a safety incident. Calendar-driven maintenance inspects everything at fixed intervals. It is easy to administer, but it spends the same effort on a fifteen-year-old pipe in dry soil as on a forty-year-old one in a flood-prone ward, and an asset that deteriorates quickly between two scheduled visits can still be missed.

The information needed to do better usually exists already. Inspection reports, repair registers, SCADA or IoT sensor feeds and weather records are all being collected, but they tend to sit in separate spreadsheets or departmental systems and are rarely analysed together. Predictive maintenance, which uses such data to estimate when a component will fail so that it can be serviced just before, is well established in manufacturing [1], [2]. Its use on public civil assets is much less developed, and reviews of the field still draw most of their examples from factories and industrial plants [3].

This paper presents the Public Infrastructure Failure Predictor, a system that brings these data sources together, estimates a failure risk for each asset, and shows the result to maintenance managers on a map. The objectives are: (i) to design a system that predicts failure risk for road segments, bridge components, pipeline sections and grid nodes from historical and environmental data; (ii) to apply machine learning classification, with time-series methods for sensor data, to estimate the probability and likely timing of failure for each asset; and (iii) to test prediction quality against historical failure records and turn the scores into a prioritised inspection list.

The contribution of this work is mainly in the integration. The individual techniques are standard; what we describe is how they are arranged into one working pipeline, from data ingestion through scoring and explanation to a dashboard, together with a prototype evaluation that shows how much a risk-ranked inspection list can improve on random and age-based selection. The rest of the paper is organised as follows. Section II reviews related work, Section III describes the architecture and methods, Section IV reports the prototype results and their limits, and Section V concludes.

 This work is licensed under a Creative Commons Attribution 4.0 International License

II. LITERATURE REVIEW

A. *Maintenance Strategies and Predictive Maintenance*

Mobley [1] describes three broad strategies: run-to-failure, preventive (time-based) and predictive (condition-based) maintenance, and argues that the third gives the best return where failure is costly and the condition of the asset can be measured. Carvalho et al. [2] reviewed the machine learning literature on predictive maintenance and found that tree ensembles and neural networks are among the most frequently applied methods. A later review by Zonta et al. [3] looked at predictive maintenance in the Industry 4.0 setting and noted that the bulk of published work concerns manufacturing equipment with dense sensor coverage. Public infrastructure differs in two ways that matter for model building: assets are spread over a large area, and sensor coverage is thin or absent, so inspection records and asset attributes have to carry much of the predictive load.

B. *Deterioration Modelling for Infrastructure*

Civil engineering has studied deterioration for a long time, mostly with statistical models. Kleiner and Rajani [4] reviewed statistical models for the structural deterioration of water mains and showed that pipe age, material, diameter, soil and break history all influence the break rate. Survival analysis, particularly the proportional hazards model of Cox [5], is a natural fit because it handles assets that have not yet failed (censored records) and produces a time-to-event estimate. More recently, Snider and McBean [6] compared machine learning with survival analysis for predicting pipe breaks, which reflects a wider question in the field: statistical models are easier to interpret, while machine learning models can capture non-linear interactions when enough records are available.

C. *Machine Learning Methods for Failure Prediction*

Random Forest [7] averages many decorrelated decision trees and is a common first choice for tabular maintenance data because it needs little tuning and tolerates mixed feature types. Gradient-boosted trees, and XGBoost [8] in particular, often give higher accuracy on structured data and are used widely in applied work. For sensor streams such as vibration or pressure, recurrent networks of the LSTM type [9] can learn patterns that unfold over many time steps. Failures are rare events, so the training data are heavily imbalanced; common remedies are class weighting and synthetic oversampling such as SMOTE [10]. Finally, a maintenance manager is unlikely to act on a score without knowing why it is high, which is the motivation for feature-attribution methods such as SHAP [11].

D. *Gap Addressed by This Work*

The studies above are largely either single-asset-class models (for example, water mains only) or industrial case studies. We found fewer descriptions of a complete system that handles several asset classes in one registry, attaches location to every record, produces an explained risk score, and presents the result in a form that a municipal engineer can use directly. The system described here is an attempt to fill that gap at prototype level.

III. METHODOLOGY

A. *System Architecture*

The system has four parts, shown in Fig. 1. Data from inspections, maintenance logs, sensors and weather sources enters through a Spring Boot backend, which registers assets, validates the incoming records and stores them in a PostgreSQL database with the PostGIS extension, so that every asset carries a geographic position. A separate Python service reads the stored history, builds features, runs the prediction models and writes the risk scores back to the database. A React dashboard then reads the scores through the backend API and displays them. Keeping the model code in its own service means the models can be retrained or replaced without touching the web application. The technologies used for each layer are listed in Table I.

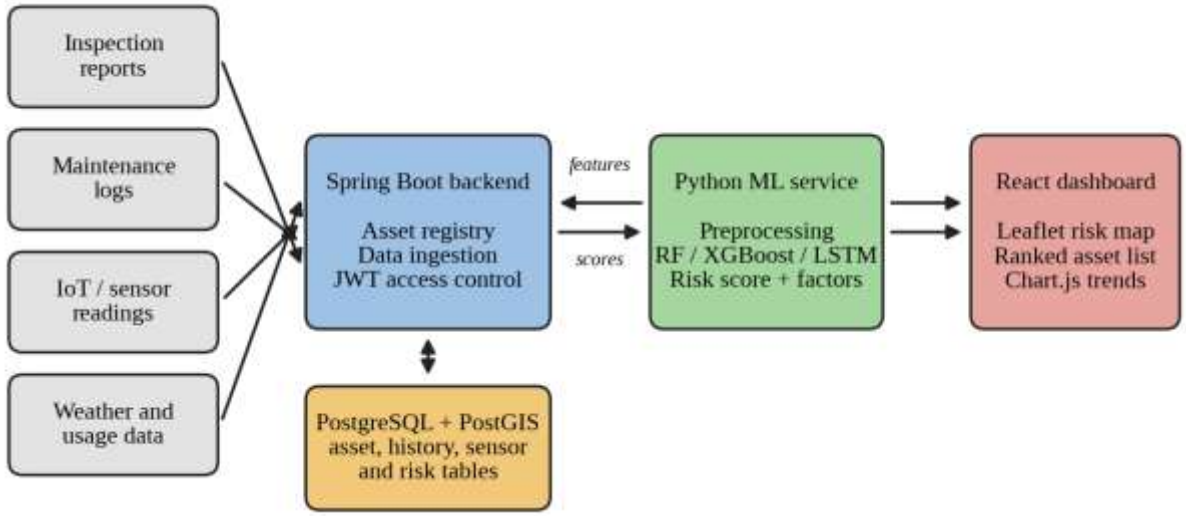


Fig. 1 Architecture of the Public Infrastructure Failure Predictor

TABLE I
TECHNOLOGY USED IN EACH LAYER

Layer	Technology	Role
ML processing	Python 3.10+, scikit-learn, XGBoost, optional TensorFlow/Keras	Feature building, model training, scoring
Backend	Java 17+, Spring Boot, Spring Security (JWT)	Asset registry, data ingestion, secured REST API
Database	PostgreSQL with PostGIS	Asset, history, sensor and risk data with geometry
Dashboard	React.js, Tailwind CSS, Leaflet.js / Mapbox, Chart.js	Risk map, ranked list, trend charts

B. *Data Sources and Feature Design*

Four groups of data are used. Asset attributes (type, material, installation year) come from the asset register. Inspection and maintenance records give the most recent condition score, the number of past repairs and the time since the asset was last inspected. Sensor and IoT readings, where available, are summarised as anomaly counts per quarter. Environmental and usage data supply annual rainfall, the number of extreme weather events and a load index such as traffic volume, pipe flow or electrical demand. Table II lists the features used in the prototype. Asset type is one-hot encoded so that a single model can serve all asset classes; separate models per class remain an option once enough real data are available.

TABLE II
INPUT FEATURES USED BY THE PREDICTION MODELS

Feature	Type	Description
Asset age	Numeric (years)	Years since installation
Usage load	Numeric (0–100)	Normalised traffic, flow or demand index
Annual rainfall	Numeric (mm)	Mean yearly rainfall at the asset location
Extreme events	Count / year	Floods, heat waves, storms recorded nearby
Days since inspection	Numeric (days)	Gap since the last field inspection
Past repairs	Count	Maintenance events in the asset history
Condition score	Numeric (1–10)	Latest inspector rating, 10 is best
Sensor anomalies	Count / quarter	Readings outside normal operating range
Asset type	Categorical (5)	Road, bridge, pipeline, grid node, drainage

C. *Preprocessing and Class Imbalance*

Missing numeric values are filled with the median of the same asset type, and a flag column records that the value was missing, since a missing inspection date is itself informative. Numeric features are standardised for the logistic regression model; the tree-based models do not need this. The target is binary: whether the asset failed within the next twelve months. Because failures are uncommon (about 5% of assets in our prototype data), accuracy alone is a misleading measure, and a model that predicts "no failure" for everything would appear to be 95% accurate. We therefore use class weighting during training and judge the models mainly by ROC-AUC, PR-AUC, recall and the share of failures captured under an inspection budget. SMOTE [10] is available as an alternative but was not needed here. In a real deployment the train and

 This work is licensed under a Creative Commons Attribution 4.0 International License

test sets should be split by date, not at random, so that the model is always tested on a period that comes after the period it was trained on.

D. *Prediction Models*

Three models were implemented in the prototype using scikit-learn [12]. Logistic regression serves as a transparent baseline. The Random Forest used 300 trees with a minimum leaf size of 5 and balanced class weights. The third model was scikit-learn's gradient boosting classifier (250 trees, depth 3, learning rate 0.05, subsampling 0.8); it stands in for XGBoost [8] in this prototype and the two share the same fit and predict interface, so exchanging them in the full system is a small change. For assets with continuous sensor data, an LSTM network [9] over a sliding window of readings is planned to produce an additional input feature for the tabular model. This part was designed but not trained, since no sensor data were available.

Hyperparameters were set once from common defaults and were not tuned on the test data. The decision threshold for converting a probability into a yes/no flag was 0.5 for logistic regression and Random Forest, and 0.25 for gradient boosting, which without class weights rarely outputs probabilities above 0.5 when only 5% of cases are positive. These values were fixed before testing.

E. *Risk Score, Tiers and Failure Window*

For each asset the model's predicted probability of failure within twelve months is stored as its risk score. Scores are grouped into three tiers for display: Low (below 0.20), Medium (0.20 to 0.50) and High (above 0.50). Since class weighting shifts the probabilities upward, the scores are best read as a ranking and not as calibrated probabilities; probability calibration is listed as future work. The expected failure window (0 to 3, 3 to 6 or 6 to 12 months) is meant to be produced by a second model trained only on assets that have failed, or alternatively by a survival model such as Cox regression [5]. This module is part of the design but has not been evaluated in the prototype, and the results in Section IV cover the failure probability and tiers only.

F. *Explaining the Risk*

Each score is shown with the factors behind it. In the prototype this is done at the model level with permutation importance, which measures how much the ROC-AUC drops when one feature is shuffled. For asset-level explanations (for example, "this pipe is high risk mainly because of its age and three sensor anomalies last quarter") SHAP values [11] will be added, since they give a contribution for every feature for every individual prediction.

G. *Backend, Database and Dashboard*

The Spring Boot service exposes REST endpoints for registering assets, uploading inspection and sensor data, and retrieving scores. Access is protected with Spring Security and JSON Web Tokens [13], with separate roles for administrators, inspectors who can submit field data, and managers who can only view results. PostgreSQL with PostGIS stores asset locations as geometry types, which allows spatial queries such as "all high-risk pipeline sections within this ward". The dashboard is a single-page React application. Assets appear as coloured markers on a Leaflet map (green, amber and red for the three tiers), a side panel lists them in descending order of risk, and selecting an asset opens its details, contributing factors and a Chart.js trend of its condition score and sensor readings over time. Filters by asset type, ward and risk tier let a manager build an inspection list for a given week.

H. *Evaluation Design*

The evaluation had two parts. The first compares the models on standard classification measures. The second reflects how the system would actually be used: a department can inspect only a fixed share of its assets in a cycle, so we measure what fraction of the assets that went on to fail were included when the inspection list was chosen in different ways. Five strategies were compared at inspection budgets of 5%, 10%, 20% and 30% of assets: random selection, oldest assets first, lowest condition score first, and the highest predicted risk from each of the three models.

IV. RESULT

A. *Experimental Data*

No municipal failure records were available to the authors at the time of writing, so a synthetic dataset was generated for testing the pipeline. It contains 6,000 assets: 1,854 roads, 1,496 pipeline sections, 895 bridge components, 882 drainage segments and 873 grid nodes. Feature values were drawn from plausible distributions, and failure within twelve months was drawn from a probability that rises with age, load, rainfall, extreme events, repair history, sensor anomalies and time since inspection, and falls with condition score, plus random noise. Of the 6,000 assets, 325 (5.4%) failed. The data were split 75/25 with stratification, giving 4,500 training and 1,500 test assets. Because the failures were generated from relationships that we chose, the results below test the software pipeline and the behaviour of the methods. They do not describe how the models would perform on real infrastructure.

 This work is licensed under a Creative Commons Attribution 4.0 International License

B. *Classification Performance*

Table III gives the results on the test set. Logistic regression had the highest ROC-AUC (0.828) and PR-AUC (0.269), followed by Random Forest (0.795) and gradient boosting (0.791); the ROC curves are in Fig. 2(a). It is not surprising that the simplest model came first here, since the synthetic failures were generated from an additive logistic relationship, which is exactly what logistic regression assumes. On real data with interacting factors the tree-based models may well do better, and this table should not be used to rank the algorithms in general. Precision is low for all models (0.16 to 0.36) because only about one asset in eighteen fails, so even a good ranking produces many false alarms at a fixed threshold. The logistic model, with class weights, found 72% of failures but at 16% precision; gradient boosting with a stricter threshold found 26% at 36% precision. Which trade-off is acceptable is a decision for the maintenance department, since it depends on the cost of an unnecessary inspection compared with the cost of a missed failure.

TABLE III
MODEL PERFORMANCE ON THE HELD-OUT TEST SET

Model	Accuracy	Precision	Recall	F1	ROC-AUC	PR-AUC
Logistic Regression	0.774	0.155	0.716	0.255	0.828	0.269
Random Forest	0.927	0.281	0.222	0.248	0.795	0.202
Gradient Boosting	0.935	0.356	0.259	0.300	0.791	0.249

C. *Inspection Prioritisation*

Table IV shows the share of test-set failures that fall inside the inspected group. With a 10% budget, random selection finds about 10.0% of failures, as expected. Inspecting the oldest assets first finds 35.8% and the lowest condition scores 29.6%, so these two simple rules are already much better than chance in this dataset. Ranking by predicted risk does better again: 45.7% for logistic regression, 44.4% for gradient boosting and 38.3% for Random Forest. At a 30% budget the best model covers 79.0% of failures, against 64.2% for the age rule. The gain over the age rule is real but moderate, roughly ten percentage points at the 10% budget, and it comes from combining several features instead of relying on one. This is the sense in which the system helps with the prioritisation problem described in the introduction. It should not be read as proof that predictive maintenance reduces failures in the field, which can only be shown by running the system alongside actual maintenance and comparing failure counts over time.

TABLE IV
SHARE OF TEST-SET FAILURES INCLUDED IN THE INSPECTED GROUP

Assets inspected	Random	Oldest first	Lowest condition	Logistic Reg.	Random Forest	Grad. Boosting
5%	5.3%	25.9%	19.8%	32.1%	27.2%	30.9%
10%	10.0%	35.8%	29.6%	45.7%	38.3%	44.4%
20%	20.1%	48.1%	44.4%	63.0%	58.0%	56.8%
30%	29.6%	64.2%	60.5%	79.0%	70.4%	67.9%

D. *Risk Tiers*

When the logistic model's scores are grouped into the three display tiers (Table V), the failure rate rises steeply from tier to tier, from 0.7% in Low to 3.4% in Medium and 15.5% in High. The High tier holds 374 of the 1,500 test assets (25%) but 58 of the 81 failures (72%), while the Low tier, a little over a third of all assets, contains only 4 failures. For a manager this means the colour on the map is a meaningful guide, even though the underlying scores are not calibrated probabilities.

TABLE V
OBSERVED FAILURES IN EACH RISK TIER (TEST SET)

Risk tier	Assets	Failures	Failure rate
Low	560	4	0.7%
Medium	566	19	3.4%
High	374	58	15.5%

E. *Factors Behind the Risk*

Fig. 2(b) shows the permutation importance of each feature in the Random Forest model. Asset age and the latest condition score were the two most influential features, followed by sensor anomalies and the time since the last inspection. Rainfall, extreme events and past repair count had a smaller effect. This ordering agrees with how the synthetic data were generated, which is what one would hope for from a working pipeline, but it is partly a consequence of the coefficients we chose and says nothing yet about real assets. The more useful result is that the explanation step runs and gives readable output that can be passed to the dashboard.

 This work is licensed under a Creative Commons Attribution 4.0 International License

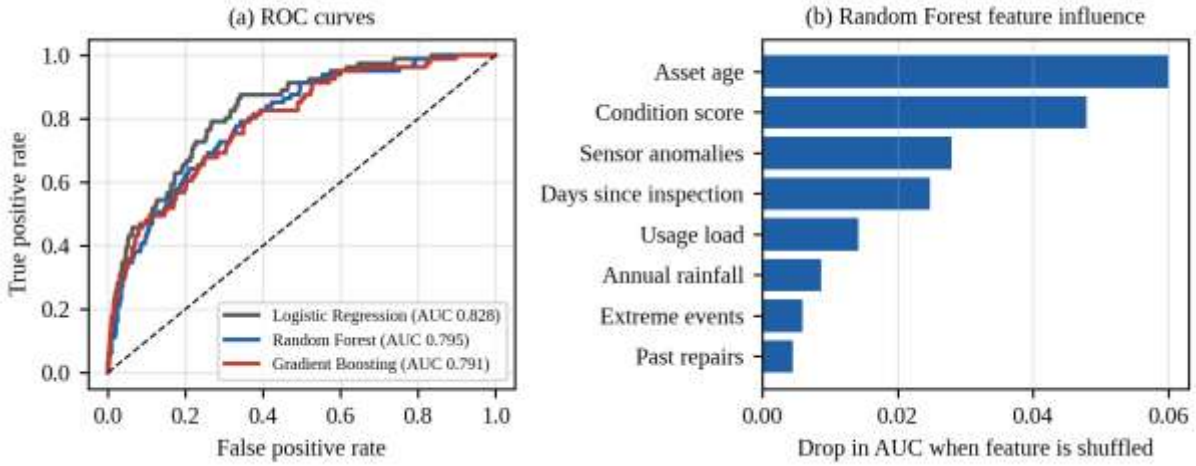


Fig. 2 (a) ROC curves of the three models and (b) permutation importance of the input features in the Random Forest model

F. *Limitations*

The main limitation is the use of synthetic data, which has been stressed above. Other limits are the following. The experiment used one random split and one random seed, so the small differences between the models (for example 0.795 against 0.791 AUC) are within what a different split could change. The scores are uncalibrated. The LSTM and failure-window modules, and the XGBoost and SHAP integrations, are designed but not yet evaluated. Finally, real failure data are subject to problems that the synthetic set does not have: incomplete registers, assets that were repaired without a record, and failures that were recorded against the wrong asset.

V. CONCLUSION

This paper described a Public Infrastructure Failure Predictor that estimates the twelve-month failure risk of roads, bridges, pipelines, grid nodes and drainage assets from their records, sensor data and surroundings, and presents the result on a map for maintenance planning. The prototype tests on synthetic data showed that the pipeline from features to risk tiers works, that ranking by predicted risk located about 46% of failures within a 10% inspection budget compared with about 10% for random selection and 36% for an oldest-first rule, and that the High tier contained 72% of the failures in about a quarter of the assets.

The next steps are clear. The most important is to obtain real inspection and failure records from a municipal body, even for a single asset class such as water pipelines, and to repeat the evaluation with a date-based split. After that, the XGBoost, LSTM and SHAP components should be completed, the failure-window model should be evaluated, the scores should be calibrated, and the dashboard should be tested with the engineers who would use it. If the results on real data are similar, the approach could help municipal departments replace fixed inspection schedules with ones that follow actual risk, which is the main aim of the project.

ACKNOWLEDGMENT

The authors thank the Department of MCA, Shri Jaykumar Rawal Institute of Technology, Dondaicha, and Kavayitri Bahinabai Chaudhari North Maharashtra University, Jalgaon, for the facilities and support provided during this project, and the project guide for advice and feedback throughout the work.

REFERENCES

[1]. D. R. Mobley, *An Introduction to Predictive Maintenance*, 2nd ed. Amsterdam, The Netherlands: Butterworth-Heinemann, 2002.

[2]. T. P. Carvalho, F. A. A. M. N. Soares, R. Vita, R. da P. Francisco, J. P. Basto and S. G. S. Alcalá, "A systematic literature review of machine learning methods applied to predictive maintenance," *Computers & Industrial Engineering*, vol. 137, Art. no. 106024, 2019.

[3]. T. Zonta, C. A. da Costa, R. da R. Righi, M. J. de Lima, E. S. da Trindade and G. P. Li, "Predictive maintenance in the Industry 4.0: A systematic literature review," *Computers & Industrial Engineering*, vol. 150, Art. no. 106889, 2020.

[4]. Y. Kleiner and B. Rajani, "Comprehensive review of structural deterioration of water mains: statistical models," *Urban Water*, vol. 3, no. 3, pp. 131–150, 2001.

[5]. D. R. Cox, "Regression models and life-tables," *Journal of the Royal Statistical Society: Series B*, vol. 34, no. 2, pp. 187–220, 1972.

[6]. S. Snider and E. McBean, "Improving urban water security through pipe-break prediction models: Machine learning or survival analysis," *Journal of Environmental Engineering*, vol. 146, no. 3, Art. no. 04019129, 2020.

[7]. L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[8]. T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, San Francisco, CA, USA, 2016, pp. 785–794.

[9]. S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[10]. N. V. Chawla, K. W. Bowyer, L. O. Hall and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[11]. S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems 30*, 2017, pp. 4765–4774.

[12]. F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[13]. M. Jones, J. Bradley and N. Sakimura, "JSON Web Token (JWT)," Internet Engineering Task Force, RFC 7519, May 2015.

 This work is licensed under a Creative Commons Attribution 4.0 International License