LIVERPOOL JOHN MOORES UNIVERSITY

Anomaly Detection using Pattern Recognition in Predictive Maintenance for Overhead Cranes

by

Shivam Goel

Supervised By: Dr Elon Correa

A Dissertation submitted to the Faculty of Engineering & Technology
Liverpool John Moors University for Msc. Data Science degree
September 2023

Table of Contents

Acknowledgement

This report is a result of my master thesis work completing the Msc Data Science program at Liverpool John Moores University the autumn of 2023.

My gratitude goes to DT Engineering in making available good quality dataset that opened avenues for deep exploration possible. Their commitment to fostering research and development is truly commendable.

Extending my appreciation to the project supervisor Dr Elon Correa for granting me with this exciting opportunity. His motivation and contributions in getting me started with the project, grasping the main ideas, prompt replies and advice about interpretation of results served as crucial checkpoints throughout the study. This kept me not only aligned with the main objectives of the case but also helped me present the results in a much more organized manner.

I am also grateful to my classmates and subject teachers for their editing help, late-night feedback, and supportive nature.

Not to mention this endeavour would not have been possible without my grandparents, parents, siblings, and friends. Their belief in me kept my spirits high and that I became capable enough.

Shivam Goel

Table of Figures

Technical Notation and Abbreviations

Terminology	Abbreviated
Crain Operation Protection Implementation	COP
Reliability Centred Maintenance	RCM
Predictive Maintenance	PDM
Artificial Intelligence	AI
Machine Learning	ML
Electric Overhead Travelling	EOT
Fast Fourier Transform	FFT
Gradient Boosting	GBM
Light Gradient Boosting	LGBM
Kaplan Meier Fitter	KM
True Positive	TP
True Negative	TN
False Positive	FP
False Negative	FN
$\frac{(\mathrm{TP} + \mathrm{TN})}{(\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN})}$	Accuracy “or” (P_o)
$\frac{\mathrm{TP}}{(\mathrm{TP} + \mathrm{FP})}$	Precision
$\frac{\mathrm{TP}}{(\mathrm{TP} + \mathrm{FN})}$	Recall
$\frac{2 * (Precision * Recall)}{(\mathrm{Precision} + \mathrm{Recall})}$	F1-score
$\frac{(TP + FP)(TP + FN) + (TN + FP)(TN + FN)}{(\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN})}$	Expected Agreement “or” (P_e)
$\frac{P_o - P_e}{1\text{- } P_e}$	Kappa Statistic

1. Abstract

Lifting-equipment specialist in the Northwest, has considered regular maintenance measure to mitigate the failures of cranes, but breakdowns continue to occur. The organization is invested in an advance AI system that accounts for data storage and presentation for the condition of the machine. The analysis is based on simulation data of 100 overhead cranes data for year 2015 to help upgrade machines used at the production site of Jaguar Land Rover. Total recorded failures for these machines were 761 throughout the year, while 2 machines experienced no failure at all during the time. The paper addresses an in-depth analysis on the sensors frequencies and how these contribute to the downtime by quantifying the relationship and machinery's health at a particular point in time. The system's architect was developed by aggregating the frequencies on a short-term and long-term basis while accounting for lags created by maintenance and errors. Ensemble-based techniques, training and testing the model on multiple dates helped achieve a robust model for real-world application in learning a machine's health, predict failure and for providing maintenance suggestions. Evaluating the model on various performance metrics such as Precision, Kappa Statistic and runtime also helped in decision making for choosing the best fit model and a reliable technique. Light gradient boosting classifier achieved over 99.5% precision as well as kappa statistic in a training time of approximately 14 seconds.

2. Introduction

2.1 Background

Solving the problem of predictive maintenance is the key to sustainability for a manufacturing unit. Suggestions mentioned in the "Safety White Paper" by Kone Cranes (2019) emphasize on the fact that a proactive provisional routine will create a flourishing workplace. Tailoring the maintenance routine as per the crane's environment, hours of operations, duty classification, local regulations, guidelines are the most important considerations if advise from manufacturer could not be obtained. Simply put, safety measure taken by expert technicians like understanding load capacity of the machinery, learning the sway time to load/unload the object appropriately, identifying the problematic area within the machinery can be made easier with safety technologies like variable speed control or remote monitoring system.

Jaguar Land Rover that uses DT Engineering Electric overhead travelling (EOT) cranes is invested in achieving excellence in operations. In the need of the hour DT Engineering is looking for an advanced version of their already existing crane operation protection (COP) system and equip it with a machine performance tracking system. This tracking system must forecast failures based on the information gathered and suggest maintenance in components like replacing engine oil, switches even lubricating wheels; Figure 1 provides the specifications of a 25-ton load capacity EOT crane. Safe to say due to the delicacy of machinery it becomes very important to dismantle the crane where the problem lies and fix it to avoid additional damage to the asset or improvise on the total downtime.

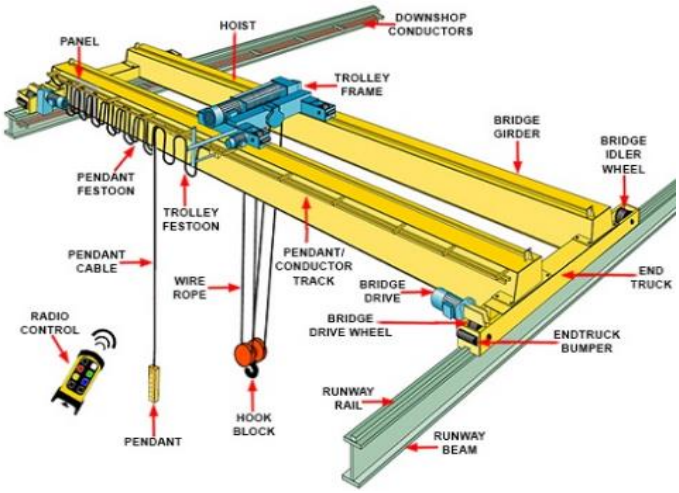


Figure 1: EOT Crane Specifications (Bhatia, 2012) in (Andreassen O.R., 2018)

2.2 Scope and Objective

Machine Learning (ML) has proven to be a go-to solution for many industries and there remain various techniques that could be deployed. So, COP V2 features may vary depending on the specific technique being followed. The project is limited to case studies of 100 cranes mentioned in the problem background. Initially, an understanding of what an overhead crane is and maintenance related whereabouts are to be learnt through descriptive and exploratory data analysis from the simulation data. There may be specific details all of which may not be possible to be solved such as distinct kinds of errors. However, the focus will remain on studying and understanding the underlying patterns in signals that lead to the malfunction of the machinery. Furthermore, it will be important to know that the maintenance is being carried out in relation to the errors and failure detected. While “time-to-event analysis” will share key insights about life span of various components, various models like Logistic regression, Gradient Boosting (GBM) and Light Gradient boosting (LGBM) will also be talked about. The evaluation of a best fit predictive model will be based on not just the accuracy but other factors such as precision and recall of model for different test periods. Consequently, various alignment and calibration techniques will help learn the model’s performance. The produced model will be expected to perform well in the real-world application and will thus be cross validated for a robust result. This study aims to enhance operational reliability by implementing predictive and preventive maintenance for overhead cranes used in the automotive manufacturing industry through AI-driven techniques.

2.3 Literature Review

Latest articles using state-of-art technologies are studied to breakdown the problem into simple steps. Various top of the line models and how can such a problem be solved where the chance of something happening is one in a thousand. The problem of pattern recognition in a time frequency plot is relevant to many domains like deciding future prices of a negotiable instrument, estimating life expectancy, and spam detection generally deploy machine learning algorithms for predictive maintenance of overhead cranes are mentioned below.

Introduction to Information Theory and Digitization 7010 Lecture 2.a refreshes the basic concepts of information theory as developed by Harry Nyquist and Claude Shannon and how they apply to digitization of data. Understand how the process of encoding lies at the heart of information theory. Calculate the entropy of various information sources. Understand the implications of the Shannon Fundamental Theorem for the transmission of information. Define and calculate the Nyquist parameters of various signals and explain the concepts of under and over sampling.

PT Bromo Steel Indonesia highlights the efficacy of PM as compared to the acting upon breakdown. The organization saved almost 1.9% of their total savings only on machine components. The study describes in detail how predictive maintenance helps minimize the damages in the machines and how the concept of predictive maintenance accelerated industry 4.0 (Hadi S., Gustopo D., and Indra D., 2019).

Förderer K.H. et al. (2015), has shone light on the key parameter involved when maintaining a crane. Types of data for semi-automatic and automatic overhead cranes include- Temperature, electric current, load weight, incidents, crane downtime, Crane mileage, velocity, acceleration, jerk etc. generated in a closed loop system. A closed loop control system learns machine's behaviour as a mathematical expression. Specific maintenance related know-hows aide in data mining is recommended. Important graphical analysis like the pareto diagram, bar-graphs and Rainflow Matrix were used to learn the condition of the machinery.

Digital Twin prediction in the Rolling Mill Stand Plant Data stored in cloud with simulated data fed by FEM simulations, Data analytics integrated with remote monitoring and augmented modules highlighted in Figure 2 is considered "principal" Solution. 6 months data was found to be sufficient for this study. Giving past values with statistical features is the key in detecting key anomalies while carefully accounting for signal correlations would account for many biases. Good to know that the working of Digital twins relies on Signal processing using Fast Fourier Transformation (FFT) and peak detection. It is also shared that the data is presently integrated to cloud and is being monitored making it an on-going process. Sotirios Panagou et al. / IFAC PapersOnLine 55-2 (2022) 132–137

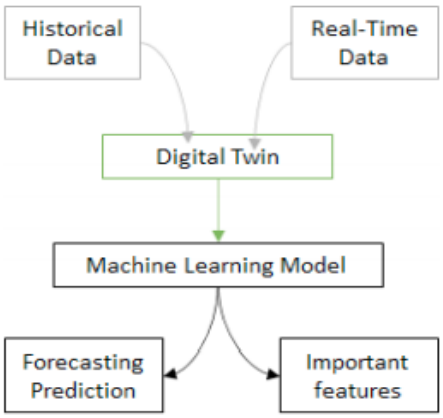


Figure 2: Digital Twins elaboration on top of ML for metadata of steel production line (Sotirios Panagou et al., 2022)

Flovik, V. (2019) carries out anomaly detection for machine learning using two separate methods for condition monitoring of the NASA bearings. Both the methods rely on the threshold value in variations of bearing with respect to the class centroid; although the methods to calculate the threshold is varied. The failure of the bearing is understood by passing a threshold value to the bearing readings as soon as the reading crosses the threshold limit, an alert that the failure is due to occur is displayed. One way of monitoring vibrations is by dimensionality reduction technique Principal Component Analysis (PCA), followed by measuring the square of Mahalanobis distance metric between clusters of normal class versus the outlier class. The alternative method covers the Autoencoder algorithm. The idea is to input the multi-variate data as windows to the model and allow it to learn itself using non-linear transformations on timestamp and generate probability distribution over K-classes using a threshold limit. Worthwhile to note that the threshold in this case is calculated using mean squared error on the loss function.

Omkar Nanekar (2022) carries out Survival Analysis effectively by successfully estimating the remaining life of the machines based on model type. This method proves to be successful in estimating the remining life based on past events and helps guide preventive maintenance strategies.

Pratik Nabriya (Jun 29, 2021) – Statistical Feature Extraction deals with time-series data in their Human Activity Recognition report by extracting 18 statistical features for each of x, y and z-axis of the gyro meter recordings using window and steps to deploy supervised binary classification techniques and achieved high performance.

TD Ameritrade (2021) on their YouTube channel clarifies how moving average allows technical analysts to understand trends in stock market. For investors it becomes important to learn the pattern to buy, sell or hold the stock although it can be difficult to determine the trend. A simple moving average is a technical indicator that tracks the security's price over a time and plots it as line. To create a moving average each date will drop the oldest record in the timeframe window and add the latest record. Important to note that a brief period moving average will capture the short-term uptrends, downtrends, and sideways trends. One thing to be aware of is the whipsaws (when the stock crosses over the moving average giving one signal and reverses quickly giving the opposite signal). Considering Long-term moving average can thus avoid the negative impact of the whipsaws. The main disadvantage of any simple moving average is their slowness to reflect price changes because each period is given equal weight, day 50 is counted equally as day 1. This creates a lag; a lag is observed when a large gain or drop hardly factors into the moving average of the whole for a while. To minimize the lag, it is recommended to use a weighted or a logarithmic moving average. These types of moving averages give recent data the most relevance and give it more weight consequently reacting quickly to security price changes. Note that moving averages do not predict future performance, they only confirm established trends.

Finally, Manani K. in their talk at PyData London (2022) explains the step-by-step process involved in Feature Engineering for time series forecasting. As advised by the speaker, to predict future retail prices by Walmart, a tabular data of features and the target is created from correlated time-series [hierarchical structure of the data; product ID belonged to a department with static features such as country of sales of the product]. A distinction between Direct Forecasting versus Recursive Forecasting is talked about and Light Gradient Boosting Model (LGBM) is showcased as top performing pure ML algorithm for forecasting due it its quick training time and specializes in dealing with correlated time series while reducing errors. Provided below is a brief distinction between the types of forecasting techniques.

- Direct Forecasting – same set of features but multiple target variables end up creating a model for each target row.
- Recursive Forecasting – Train the data once to predict one step ahead forecast [for time; t+1], the model then takes the forecasted value and adds it to the training set and trains again for t+2 time. The process continues till the test is complete. Note multi-step forecasting technique takes seperate set of features for each prediction it makes. Recursive forecasting is often used in time series forecasting, such as ARIMA or state space models.

An important takeaway here is that LGBM itself is not inherently a recursive forecasting model. In the context of PDM using LGBM, one can choose how the data will be structured for training.

3. Data Acquisition and Description

The Foremost step in Reliability Centred Maintenance (RCM) discussed during the Proceedings of the 2022 winter Simulation Conference suggests that Simulation data is better for predictive analysis for numerous reasons. Figure 3 shows the trend of synthetic data and how it is believed to overshadow the use of real data in the upcoming future. For this case, machine specific material is made by the manufacturer and contractor DT Engineering and is supplied to Jaguar Land Rover. Simulation Data for use and equipment history is supplied as well by DT Engineering. Other cranes used by other operators are not considered.

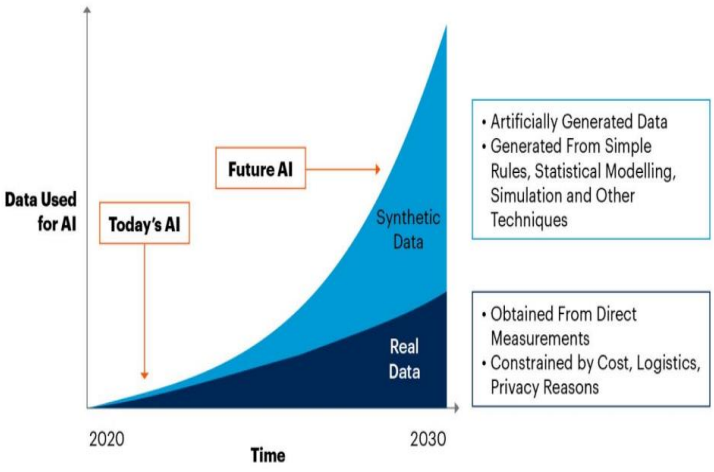


Figure 3: Need of Simulation Data (Gartner, 2021) in (Burger et al, 2022)

Unfolded by Nanekar O.(2022), The simulation dataset was earlier provided by Microsoft Azure specially for Predictive Maintenance on Kaggle but was removed on 15th October 2020 for organizational purposes. For the case 5 ".csv" files were obtained in a zip folder listed below.

"PdM_machines.csv" contains the metadata of the machines. This descriptive information includes machine ID, machine model (4 types) and age (in years). Safe to say that the cranes are known to have a load capacity of around 35-45 tons, but this information is not being considered for the case since all the machines are believed to be of equal duty. Dimensions of the file: 100 rows x 3 columns. Below are the total occurrences of various "model":

1. model1: 16
2. model2: 17
3. model3: 35
4. model4: 32

Not a huge imbalance is seen here and can be presumed to be normally distributed.
"PdM_telemetry.csv" – contains the sensors readings taken sequentially on an hourly basis for the year 2015-16 for respective Machine Id. Nevertheless, sensory data include volt (in volts), rotate (Rounds Per Minute), pressure (in psi), vibration (in mm/s).

	datetime	machineID	volt	rotate	pressure	vibration
876099	2016-01-01 06:00:00	100	171.336037	496.09687	79.095538	37.845245

Table 1:Tabular Telemetry Obtained

As seen in Table 1 as the tail of tabular timestamp, total instances equal 876100 while independent variables are 6 before structuring the data. Worthwhile to know that the data is in fact hierarchical in nature that means for each machine ID the timestamps are concatenated horizontally for the 4 mentioned sensors. This structuring of data is needed to restore the information while removing noise. This is a regular snapshot of each machine and carries a relation to itself (Autocorrelation). Additionally, the importance of time series data is related to causality which will be further uncovered in Survival Analysis.

"PdM_errors.csv" file – includes details about the type of error that occurred in respective machine_ID with details about time of occurrence. Errors here are any technical blockade in the readings of the sensors. For instance, in an infrared sensor dust hinders the reading, a specific type of error is observed. Due to sensitivity wrong information is passed. It shows a glitch in the algorithm, but it is not to be confused with the failures since these do not result in shutting down of a machine. Generally, we won't know the different types of errors but still try and find patterns with signals data. Various types of errors are recorded. Frequency of distinct error types:

1. errorID1: 1010
2. errorID2: 988
3. errorID3: 838
4. errorID4: 727
5. errorID5: 356

While the original shape of the dataset contains 3919 entries and 3 fields.

"PdM_maint.csv" : This set of data carries information about the regularly scheduled replacements of components (preventive maintenance). Any rectifications made in belt, oil, engine, or filters etc. for each machine_ID are provided in this. 3286 instances with 3 columns are the dimensions of this dataset. Find the frequency of replacement for each component below:

1. comp1: 804
2. comp2: 863
3. comp3: 808
4. comp4: 811

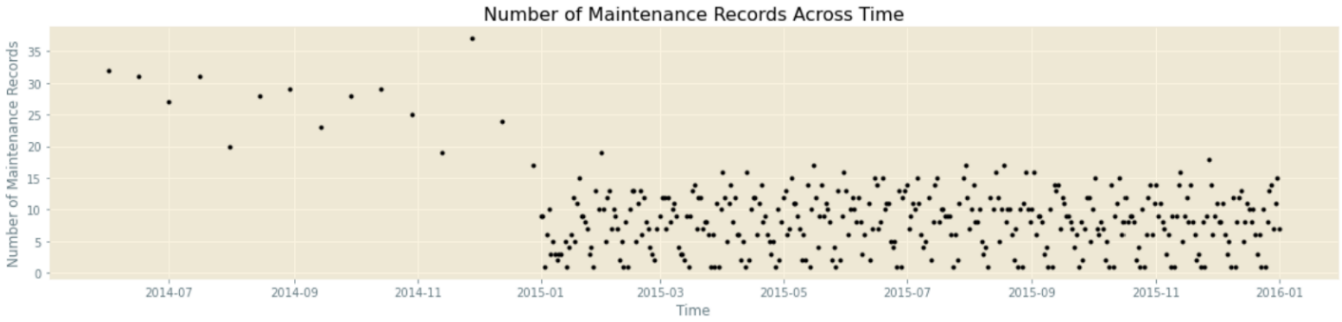


Figure 4: Number of maintenance records

Scatterplot in Figure 4 provides a clearer picture of the entries made across time. Notice that a lot of maintenance records are captured prior to the year 2015. Although these records could be used

to draw any correlations with failure or errors, the main objective is to observe any deviations in the frequency of sensors and because no such data for the year 2014 is acquired hence these rows will eventually be dropped before carrying out any machine learning algorithms. In addition, it could be worthwhile to scrutinize whether the maintenance also is being carried out after an error is observed.

"PdM_failures.csv" : Information about component replacement due to failure is recorded in this file. This datafile is thus a subset of the maintenance data since it becomes a reactive maintenance record. The shape of the dataset consists of 761 instances and 3 attributes namely "datetime", "machine_ID" and "failure" [Target column]. The failure column being categorical in nature contains multiple classes . Total occurrences of failures recorded are as followed:

1. comp1 : 192
2. comp2 : 259
3. comp3 : 131
4. comp4 : 179

Descriptive/explanatory analysis on maintenance data separately. 2014 needs to be dropped for modelling to remove any incomplete information. Better to not completely ignore it though.

Note: no null values or duplicate records found in the mentioned 5 files.

4. Methodology

A view about the techniques is disclosed in this section.

4.1 Environment Setup

This section aims to provide an overview of programming languages and analytics platforms used for this case study.

While every analytics framework has its own rewards, two major platforms were strategically decided upon. To begin with, a Visual Studio Code (VS Code) file that supports Jupyter notebook running on pre-installed python version 3.9.7 was utilized. Code editing capabilities and help commands in VS Code with in-built runtime tracking facilitates a dynamic data manipulation environment. Simultaneously, Google's Cloud emerged as a popular platform for machine learning tasks. Its integration with a free access T4 GPU for reduced computation time and default execution time monitoring facilitates accelerated model training and experimentation. Therefore, these two platforms were chosen. The Python-specific environments, both hardware based, and cloud based ensured seamless transition between various stages of the project and ensured integrity within the operations.

packages and libraries will be imported to explain the system and boundaries to the model for Data preparation and formulating the surroundings mathematically. Table 2 shows the necessary packages and their version. Learning the version becomes important when setting up the environment because there may be packages that only support certain version. For instance,

package named “anai” is an automated machine learning library for tabular data and showcased robust results on Kaggle but is sensitive to versions of libraries like scikit-learn and Numpy.

Library	Version
Python	3.9.7 (default, Sep 16 2021, 16:59:28) [MSC v....
Pandas	1.3.4
NumPy	1.20.3
Scikit-Learn	1.2.0
Gradient Boosting (from scikit-learn)	1.2.0
Logistic Regression (from scikit-learn)	1.2.0

Table 2: Software Versions

After installing libraries, read in, validate, and clean the data. Consider whether the machine readability of data and metadata could be improved. Visualize multi-variate relations and autocorrelations in sensors. Check for missing values, type of distribution, outliers, skewness, kurtosis. Identify seasonal trends in the sensors records.

4. 2 Underlying Assumptions

- When dealing with time-series, it is essential to determine if the time-series is regular or irregular. For this case, the data points captured are assumed to be rounded to the nearest hour. Although maintenance, errors and failures are unique events that act as hazardous functions to be recorded at irregular intervals, “PdM_telemetry.csv” is regular in nature.
- Other assumptions include- Each sensor is independent of each other.
- The Time-series follows an autocorrelation meaning recent time observations are more likely to affect the future happenings rather than the former ones. In other words, if a machine sees a problem in the beginning of time frame, then the future happenings are deemed to face issues as well.
- The performance of a machine depreciates with time as it performs its daily operations.
- Failure depends on sensors recordings.

4.3 Problem Identification

<u>Dealing with Time-Series</u>

- Most telemetry experiences repeating pattern over time. This could be the weekend offs, union strike, yearly bank holiday etc. While this is not an assumption, a check for seasonality must be taken into consideration.
- Another importance of time series data is related to causality. If a quantity varies with time, then some effect has caused that change. Although there may be scenarios that a cyclical or repeatable time series informs one about the nature of the system.
- Check Autocorrelation to scrutinize which lags are the most important. {Autocorrelation tells us how much a time-series is related to different lag values of itself.} – Relevant to

the concept of distributed lengths. – Thus, feature selection is usually done after predicting and checking feature importance.

Visualizing the sensors records and changing number of peaks, Inter-Quartile Range (IQR), min/max etc. will describe the usage of machine and the environment setup to the model.

Normality Check & Correlation

Initially a thorough descriptive analysis needs to be done to understand what each dataset contains and how could a final data frame be created.

normality of sensors will be checked using a QQ-plot (theoretical values parallel to the observed values). This will guide the appropriate model and techniques to use. Note if the data is not normally distributed then various non-parametric measures may need to be considered. Moving further, As observed there are several categorical variables and drawing correlation of them with failure (Target variable) also needs to be confirmed. This will help consider appropriate maintenance solutions. For instance, if Model type does not play any importance in influencing failure, then we may as well plan to keep it or drop it depending on the type of algorithm going to be applied. Worthwhile to note that such correlations need to be considered when trying to learn more about the performance of the model. Followed by deviations in frequency and Kaplan Meier Fitter to get a basic idea about each component's general life span to add on to the prudence planning.

Time-Series Analysis

Based on literature review, time series that depend on recent values affect the future readings is considered to follow an autocorrelation. Consider this test to be a comparison of time with itself when the whole duration is divided in distributed lengths. When such autocorrelation is followed, a good approach in machine learning would be to give the model most recent values to predict the future values. An example would be to predict the future one-month readings based on recent three months readings. Here lags play an important role and one must be careful in how many lags to consider. The concept of lags can be better understood in Figure 5. The snippet is taken from Wikipedia is a representation of stocks taken on 8 hours basis. Static Average shown (in Green) frequency with its interquartile range, Simple Moving Average (in Blue) and Exponential Moving Average (in Gold). Notice how a significant drop in the frequency cannot be seen so easily in static averages whereas, when compared to exponential moving average, the model seems a better fit with the overall curve of the real fluctuations.

Figure 5: Simple Moving average v/s Exponential Moving Averages v/s Static average (Wikipedia , https://en.wikipedia.org/wiki/Moving_average)

As in the literature review there are many approaches of applying a machine learning process in predictive maintenance, depending on the origin of the data and the motive of the analysis.

4.4 Proposed Algorithms

Logistic Regression: This ML algorithm uses an "S" shaped logistic function that estimates the probability of an event occurring or not, using the input data. The statistical test mentioned in Figure 6 on the right shows an example of the working of logistic regression; when the threshold is set to 0.5, the values below the setpoint will all be predicted as 0 whereas, for points above the threshold limit will be considered 1. Primarily determining whether there is any failure at all or not. The sigmoid activation function used in the supervised classifier algorithm makes it a good start point.

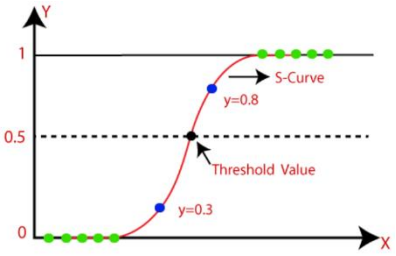


Figure 6: Binary Classification using Logistic Regression

Other models are based on the techniques taught by Ashish Patel and Kishan Manani. Due to the popularity of ensemble ML techniques and ease of implementation of LGBM will be talked about and compared with its competitor Gradient Boosting Classifier to pick the best fit model. To

understand Both the GBM and LGBM it becomes helpful to learn the workflow of a decision tree based on which the two gradient boosting models operate.

Decision Trees: Binary trees that recursively split the data until we are left with pure leaf nodes that contain only one type of class. When data is not linearly separable, the data can be split into parts using different combinations of condition splitting. A decision node contains a condition to split the data while a leaf node helps to decide the class of new data point. The need of ML arises to learn the best splitting condition (features, where most pure leaf nodes are attained). Thus, a decision tree classifier transverse across every feature and feature value to search for the best corresponding threshold for multiple splits. Figure 7 shows an example of how pure leaf nodes in decision trees are attained. (Educba, 2023)

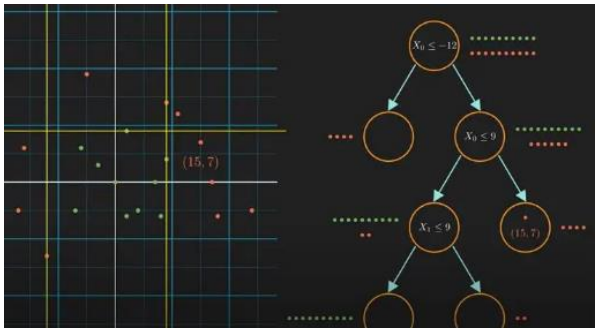


Figure 7 Decision tree condition split pure leaf nodes (available at: https://www.educba.com/decision-tree-types/)

Gradient Boosting: An ensemble-based algorithm that unifies several weak decision trees (models) and builds a stronger predictive model in a stage wise manner that rectifies the errors of the preceding model. This model iteratively gathers gradient information (rate of change of a function with respect to its input) to determine the magnitude and intensity of adaptations to consider at each step. In other words, the gradient of the loss function with respect to the predictions of current ensemble explains the adjustments to be made to minimize overall loss.

Light Gradient Boosting: This type of predictive analysis follows the Gradient boosting method of combining the weak models and optimized for exceptional speed and efficiency. Making use of a histogram-based approach to binning data, faster training time and reduced memory usage for big data sets show promising results.

4.5 Workflow

As per the literature Review, excruciating digging has been done on feature engineering and data preparation. The first method follows a generic ML routine where the numeric columns must be scaled, and categorical columns must be encoded, while balancing the target variable. Note, static aggregate values will be calculated. The second approach would be to follow the approach shared in PyData 2022 London talk for dealing with correlated time-series. Good to mention that the Logistic Regression classifier was used to check the quality of data preparation for both the methods and based on the results a more advanced ensemble-based classifier will be deployed and compared. Both the methods of data preparation are described in detail below.

I. Method 1:

- ✓ Grouping the telemetry dataset based on Machine ID and date to get daily aggregated values.
- ✓ Compute aggregated Static Features of sensors and Fourier transformed sensors readings.
    - Included Features: min, max, mean, median, count, IQR, skewness and kurtosis.
    - Mean Absolute Percentage Error (MAPE), Root Mean Squared Error (RMSE) and energy ratio for main sensors records (volt, rotate, vibration and pressure) are also computed.
    - Please refer to the appendix to get a better understanding on the formulas to calculate the Fourier transformed values, MAPE, RMSE and energy.
    - Final structure of the "aggregated data" consists of 36600 rows and 74 columns.
    - Note Various different categories were given to the model however giving in more than 100 columns resulted in over-fitting when compared the test set to validation set. Thus the final structure may vary depending on the input variables provided.
- ✓ Once the aggregated metadata is attained, the rest of the files are merged one-by-one using looping commands based on machine_ID while dropping any duplicate rows. Good to know that the value counts for different classes in various columns may vary after merging but this should not be of concern as the program tries to preserve most of the information/
- ✓ Label Encode the target column (failure column) and dummy encode other object type columns.
- ✓ Normalizing the continuous columns using various scaling techniques.
    - Standard Scaler: Centres and scales input features so they have a mean of 0 and a standard deviation of 1, often used to make algorithms converge faster.
    - Robust Scaler: Scales features using the median and the interquartile range, making it robust to outliers compared to Standard Scaler.
    - Min/Max Scaler: Rescales the data to a specified range (usually [0, 1]) by subtracting the minimum value and dividing by the range, helping in normalizing the feature scales.
    - Max Abs Scaler: Scales each feature by its maximum absolute value, which is meant for data that is already centred at zero or sparse data.
    - Normalizer: Scales individual samples to have a unit norm, which is useful for vector space models and is applied on a row-by-row basis, not based on the dataset features.

- ✓ Train/Test Split
    - Splitting is done using "sklearn.preprocessing" "train_test_split" library based on percentage.
    - 2o% of the total data is taken out as the test set.
    - The remaining 80% of the data is again partitioned by 18% becoming the validation set for cross-validation and the rest as train set on which the model learns (trains).

II. Method 2: This method of data preparation follows a more challenging yet reliable solution to the problem. As mentioned by Manani, K.(2022) trend can be captured better using simple moving average. must be taken on a rolling basis and recursive forecasting

technique used by gradient boosting will be followed. Take rolling mean and standard deviation capturing short term variations and long-term variations (3 hours and 24 hours in this case). Feature Engineering refers to manipulating various data sources to create attributes which unveils the crane's health status at any point in time. Owing to the properties of respective sources of data, features are reformatted for better interpretability.

- ✓ Lag Features from Telemetry
  - A popular approach would be to pick a window size to get the lag features by computing rolling central tendency measures such as mean and standard deviation to capture the short-term fluctuations of the Cranes over the lag window. Simple moving average and standard deviation of the telemetry data over the last 3-hour lag window is calculated for every 3 hours. To account for a longer duration, 24-hour lag features are calculated as well. One might be curious to why 3 hours rolling windows. After viewing several literature and evaluation of test results with data aggregated only based on 24 hours data leakage was observed. This data leakage arises because failure events are already very less compared to normal state records and thus to build a model around it, several machines' failures records have to be considered and the sensors recordings will be aggregated from seconds originally to daily basis. Note Aggregation on 3 hours to 6 hours are recommended to capture the short-term fluctuations.
  - After concatenating "3 hours" calculated table and "24 hours" table based on column, the dimension of the data frame named "telemetry_feat" is 291300 rows and 18 columns.
- ✓ Lag Features from Errors
  - Another influence on failures could be from number of errors observed. Here, error IDs are object type and thus cannot be averaged over time periods like the telemetry measurements. Instead, counting the occurrence of each error type in the specified lagging window could turn out to be an effective way. This is achieved by formatting the error data to have one observation for unique machine ID per unit of time at which at least one error occurred. For timepoints taken every three hours, the total number of various types of error over the recent 24 hours in calculated.
  - Summary table for the lags in errors in provided in table below:

	machineID	error1count	error2count	error3count	error4count	error5count
count	291400.00000	291400.000000	291400.000000	291400.000000	291400.000000	291400.000000
mean	50.50000	0.027649	0.027069	0.022907	0.019904	0.009753
std	28.86612	0.166273	0.164429	0.151453	0.140820	0.098797
min	1.00000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	25.75000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	50.50000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	75.25000	0.000000	0.000000	0.000000	0.000000	0.000000
max	100.00000	2.000000	2.000000	2.000000	2.000000	2.000000

Table 3: Daily counts of errors for each machine ID.

- ✓ Last Replacement of component in days
  - Maintenance records play a pivotal role and must be dealt with carefully. Unlike the telemetry and the errors data, getting lags from maintenance is a bit more complicated. An interesting feature that could be calculated here would be to get the count of replacements of each component in the last quarter to take in consideration the frequency of replacements. Although, a better approach to this column would be to tell the model exactly how many days have passed since the component was last replaced as that would be expected to correlate better with component failures. The longer a component is used without replacement, the more wear-n-tear should be expected.
  - The features from this data are worked out in a more custom way. This type of improvised engineering of object type features can play a key role in predictive maintenance. Table 4 below shows the summary of components last replacement table.
  - To get these values, maintenance column was grouped based on machine ID and datetime to get the aggregated sum after dummy encoding the dataset. Later the difference in datetime is anticipated. See codes in Appendix B for the same.

	machineID	comp1	comp2	comp3	comp4
count	876100.000000	876100.000000	876100.000000	876100.000000	876100.000000
mean	50.500000	53.525185	51.540806	52.725962	53.834191
std	28.866087	62.491679	59.269254	58.873114	59.707978
min	1.000000	0.000000	0.000000	0.000000	0.000000
25%	25.750000	13.291667	12.125000	13.125000	13.000000
50%	50.500000	32.791667	29.666667	32.291667	32.500000
75%	75.250000	68.708333	66.541667	67.333333	70.458333
max	100.000000	491.958333	348.958333	370.958333	394.958333

Table 4: Summary Table of Maintenance Data after getting dummies and on calculating the lags.

- ✓ Machines Data
  - There were no modifications made to the machines data. Although not mentioned in the dataset, however if the age was recorded as first day of service, this will have to be changed to total numbers of service in years to get a numeric value.
- ✓ Final feature engineered matrix is obtained by merging all the above tables. Shape of the data should be 291,341 records and 30 columns. Note this does not include the "PdM_machines.csv". The same is dummy encoded when creating train and test split.
- ✓ Train/Test Split
  - Time-based cross-validation approach also referred to as "rolling" or "expanding" window validation using threshold dates to evaluate the model against multiple time

periods. This strategy will be particularly useful for time series data where the temporal order of observations matter.

- Why Threshold Dates?
  - The threshold dates help simulate a real-world scenario where the model trains on historical data and test its performance on future data. This approach assesses how well the model generalizes to unseen future events. – based on empirical tests and general train/test split threshold dates are created.
- Why Three Pairs of Dates?
  - Using multiple pairs of threshold dates allows you to assess the model's performance on different time periods. This can help one evaluate the model's stability and robustness across different stages of the dataset.
- What Happens Between the Dates?
  - The data between "last_train_date" and "first_test_date" is neither used for training nor testing in the current loop iteration. It is reserved for a different fold in cross-validation or another iteration in a similar time-based.

Machine Learning workflow for different datatypes. (PyData London, 2022) and comparison of both methods.

	ML for Regression/Classification	***ML for Forecasting***
Train/Test split	(Split by percent (20%) to get test set, further split train set by 18% to get a validation set.	Split based on time. Multiple splits to test on different time periods; no validation set.
Features and Target	Pre-compute features and target before prediction	Target is a specific point in time. Features built from target at predict time.
Model Prediction	Only trained model needed to predict	Require trained model and training set
Feature engineering	Static central tendency measurements till 4^{th} order like min, max, median, IQR, skewness, Kurtosis of Fourier transformed readings, mean absolute percentage error (MAPE), Root Mean Squared Error (RMSE) and energy.	Intelligent feature engineering and data leakage issues, consider simple moving average with lags, exponential smoothing, and set threshold dates (holidays) for detrending and seasonal decomposition of the timestamps.
Encoding categorical columns, and Scaling/Normalizing numeric columns	Dummy Encode and Convert to Numerical type. Scaling techniques like min/max scaler, standard scalar, and robust scalar and more, while label encoding the target becomes essential.	• Dummy Encode and Convert to Numerical type; gather useful information such as daily count of error types per machine ID, days since last maintenance of each component etc. • No Normalization or label encoding required.

Performance Metrics	Classification Report, Accuracy, Precision, F1 recall, Random Guess Accuracy, Kappa, Macro average	Same applied

5. Results

This section aims to provide the output generated and the insights found from the analysis.

Data Analysis

Figure 8 below can be viewed to check the distribution of failure records based on unique Machine IDs across the whole period. This confirms that the failures have a random chance of occurrence and are distributed randomly across machines. ID 6 and ID 77 experienced no failure in the period and hence can't be seen in the plot.

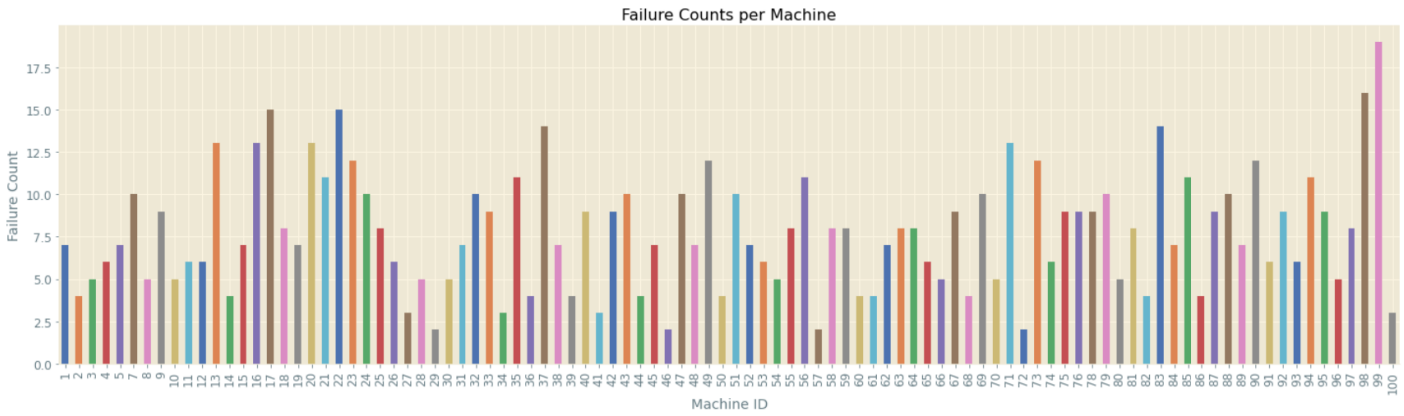


Figure 8: Total Number of Failures Recorded for each Machine ID across during the period.

Because Machine ID 99 experienced the greatest number of failures, a train set for exploration was created to check for any outliers in telemetry.

The frequency plot of machine ID 99 in Figure 9 for the first two months presents a better understanding to the problem that is being solved. The significant deviation in sensors records for a certain period can be considered as abnormal behaviour in the performance of the machine. The objective when developing a machine learning model Therefore should account for such deviations in the long run. When the readings go 3 standard deviations above or below the average will be considered as anomalous in nature, this is when signs for maintenance are observed.

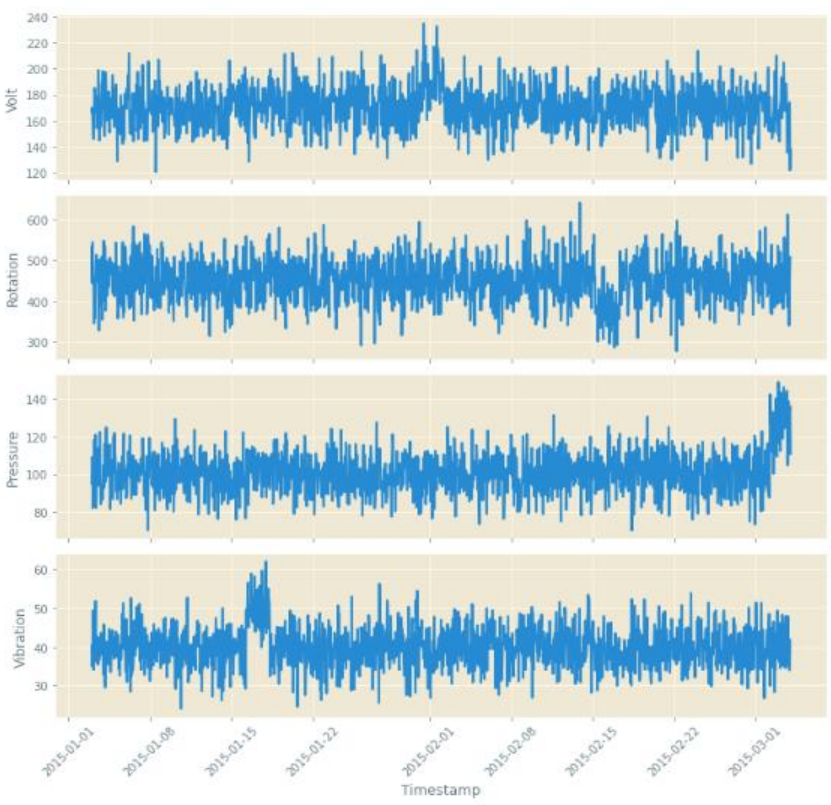


Figure 9: Frequency Plot for Machine ID 99 for first 2 months

Now to check for normality of the telemetry data a QQ plot statistical illustration will be conducted. Figure 10 makes clear of any outliers present in the data. The graphical representation is a statistical test for check of the gaussian distribution. The Theoretical values are expected to align (be parallel) with the observed values when there are none unexpectedly high or low readings present. Since, the values do not overlap on each other, The test confirms the presence of outliers in the sample data.

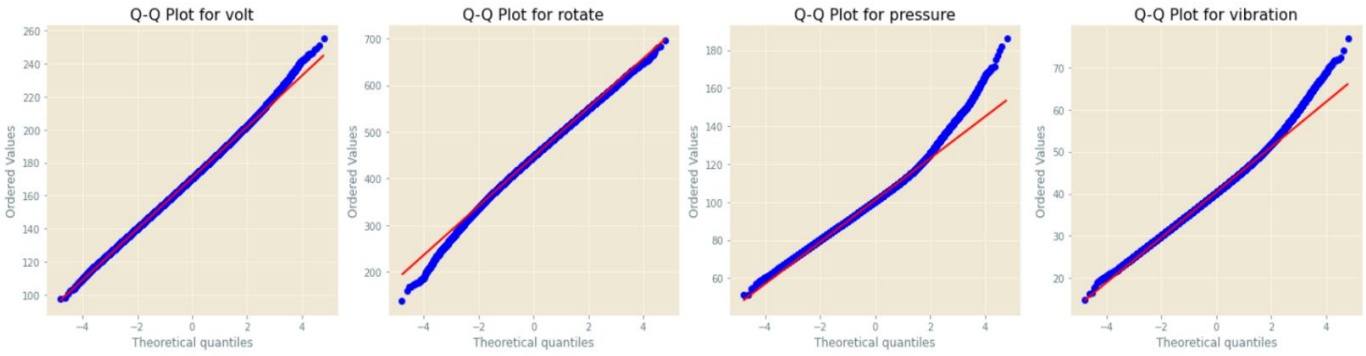


Figure 10: Normality Check Plot (QQ Plot)

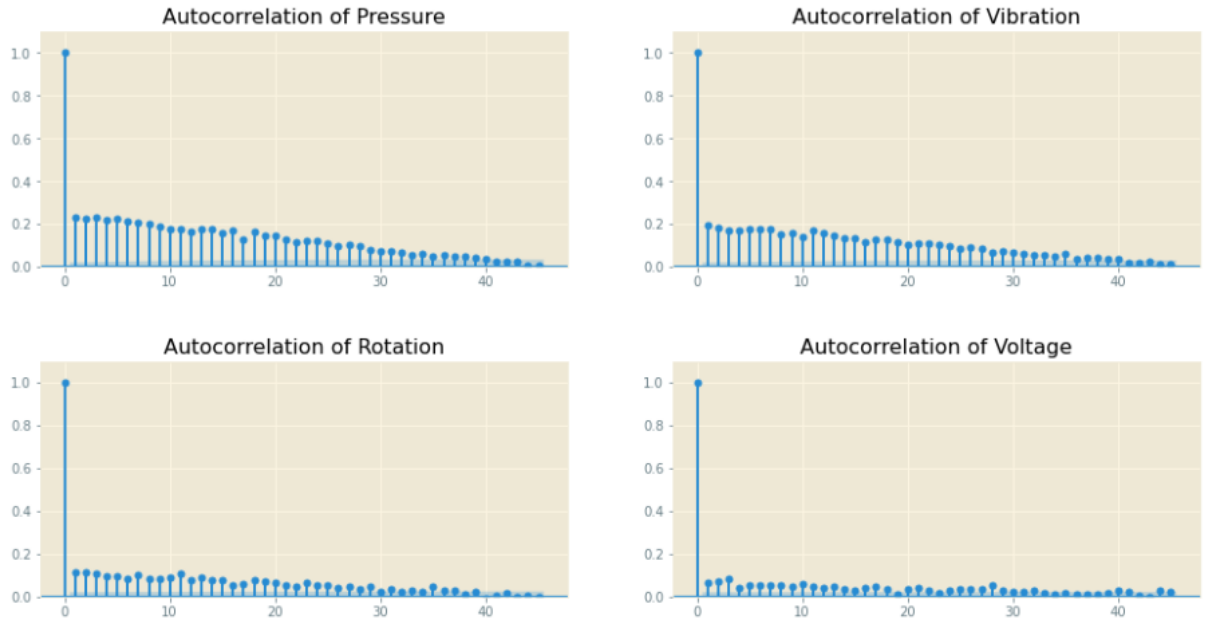


Figure 11: Autocorrelation for Machine ID 99

Figure 11 depicts how dependent each of the sensor readings depend on their recent hours for machine ID 99. Pressure recordings depend of readings of past 30 hours approximately while voltage does not depend on what happened in the past.

Survival Analysis

Moving forward, the results for Kaplan Meier Fitter (KM) will complement the study of general life span of various components within different models. Figure 12 represents a visual representation of a subject's survival rate across time.

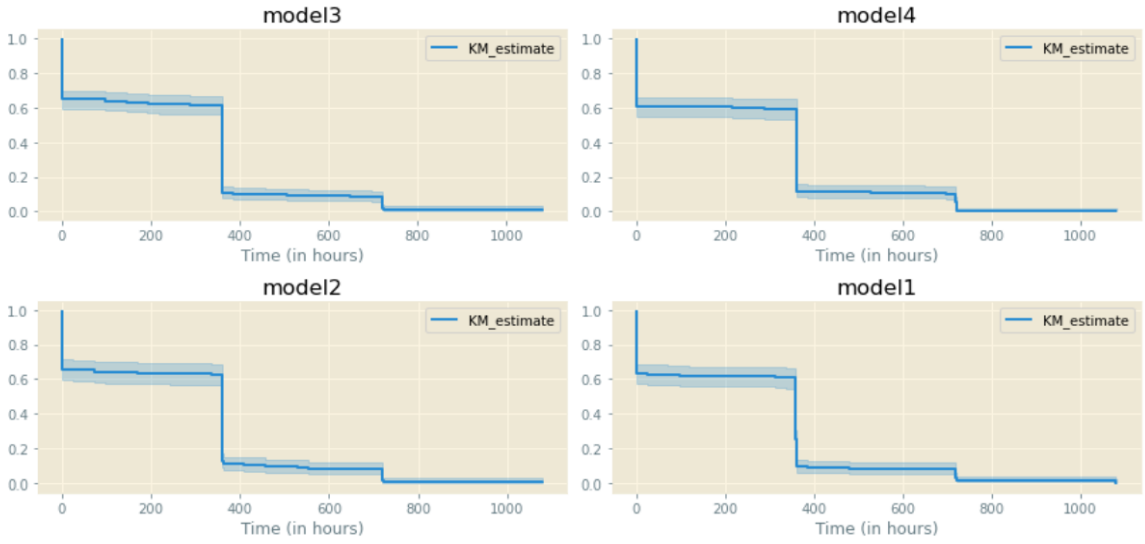


Figure 12: Life estimation using Kaplan Meier Fitter for Component 1 based on model type.

In the figure not much, noticeable difference was observed for component 1 in any model. This means that around 50% of component 1 die within 380 hours of replacement no matter which model they belong to. Similarly, life cycle of other components could also be found. In the dynamic environment of operations and logistics, setting a provisional schedule can be painful while this approach may seem handy in many cases. The technicians can plan for the replacements a few weeks in advance of such components to avoid carrying out last day deliveries and mitigate downtime.

This seems enough information to continue developing the AI system.

Model Evaluation

Evaluation Metrics

1. *Confusion Matrix:* A table used to describe the performance of a classification model on a set of data for which the true values are known.
2. *Accuracy:* Proportion of correctly predicted observations to the total observations.
3. *Precision:* The ratio of correctly predicted positive observations to the total predicted positives. The questions it answers is of all machine IDs that labelled as survived, how many survived? High precision relates to the low false positive rate.
4. *Recall (Sensitivity):* The ratio of correctly predicted positive observations to all observations in actual class. The question recall answers is: of all the Machines that truly survived, how many did we label correctly.
5. *F1-Score:* The weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account.
6. *Macro Precision, Recall, F1:* The macro average computes the metric independently for each class and then takes the average treating all classes equally.
7. *Micro-Averaged Precision/Recall/F1:* Calculate metrics globally by counting the total true positives, false negatives, and false positives.
8. *Average Accuracy:* This is the average of the diagonal in the one-vs-all confusion matrix.
9. *Majority Class Accuracy, Recall, Precision, F1:* These are the accuracy, recall, precision and F1 score of the majority class (the class with the most samples).
10. *Expected Accuracy:* This is the accuracy that would be achieved by a random model.
11. *Kappa Statistic:* This is a statistical test that measures inter-rater agreement for qualitative (categorical) items. It is generally thought to be a more robust measure than simple percent agreement calculation, as κ considers the possibility of the agreement occurring by chance.

	Scaler	Accuracy	Precision	Recall	F1
0	StandardScaler	0.989186	0.756306	0.691556	0.715850
1	RobustScaler	0.989755	0.765934	0.717849	0.735993
2	MinMaxScaler	0.990324	0.852964	0.638035	0.720454
3	MaxAbsScaler	0.989945	0.826008	0.639356	0.713293
4	Normalizer	0.989186	0.866675	0.562190	0.668281

Figure 13: Logistic Regression Test results using Method 1

	Scaler	Accuracy	Precision	Recall	F1
0	StandardScaler	0.979000	0.807659	0.639016	0.708397
1	RobustScaler	0.979024	0.808724	0.637443	0.707885
2	MinMaxScaler	0.979119	0.821803	0.624709	0.703779
3	MaxAbsScaler	0.978476	0.839936	0.582208	0.677672
4	Normalizer	0.964150	0.243130	0.208393	0.211091

Figure 14: Logistic Regression Test result using Method 2

Figure 13 and Figure 14 shows the test results for the logistic regression model for both the methods created. Although first method shows slightly better results in terms of accuracy, recall and F1 score compared the second method, precision was higher for the second method of preparation. Furthermore, it may be worthwhile to note that both the models experienced higher scores on the test set as compared to the validation set which confirms that there was no overfitting. Another observation is that Min-Max Scaler seems to be a good way of scaling the data.

Some of the other hyper-parameters taken into consideration here were the regularization method "L2 Regularization" also known as Ridge Regression. This will enable the model to minimize overfitting the model when high collinearity is observed between predicted variable with its predictors. Furthermore, to deal with class imbalance many techniques like under-sample, over-sampling and assigning class weights becomes essential. While class-weights needs to be adjusted manually, under-sampling technique showed low results as majority of information was lost in reducing the majority column equivalent to the minority class. The number of iterations also had to be increased from 1000 being default to at least 7500 for the model to complete the training phase at a learning rate of 0.1.

Looking at the results, one may assume that the first method of data preparation is better, however that is not the case when an LGBM model is applied to the former method of data preparation. Below figures in an example of information received after setting the "verbose" to 2 to be able to read and examine how well the model is converging during the training phase. In this case LGBM is unable to converge properly and was observed to converge towards infinity raising a value error key. Figure 15 displays the verbose results for LGBM developed using the first method. Observed clearly, the model finds it difficult to split the nodes properly based on the given information. This could be due to the fact there was data leakage when only taking in consideration the long-term effects of the sensors. Not only that, the train-test split was also done on a percentage split without giving a gap of time between the

```
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 10
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 11
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Debug] Trained a tree with leaves = 13 and depth = 8
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Debug] Trained a tree with leaves = 18 and depth = 9
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 10
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 9
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Debug] Trained a tree with leaves = 11 and depth = 8
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Debug] Trained a tree with leaves = 17 and depth = 8
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 12
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 13
[LightGBM] [Debug] Trained a tree with leaves = 31 and depth = 10
```

Figure 15: verbose of LGBM (Method 1)

“last_train_date” and “first_test_date” which could have potentially taken a part of test set into the training set. This gave rise to the second method of pre-processing and the adopted secondary method of splitting as well. Figure 16 on the contrary is the verbose for LGBM using the secondary adopted method.

The training results seem much more likely when the training set input is of good quality. This sets the base to be using rolling aggregate functions. Note that the gradient boosting classifier and light gradient boosting classifier do not require any scaling to be considered. These models deal with collinearity effectively as well.

For the study both GBM and LGBM use default hyper-parameters and combine the weak learner. Therefore, dealing with imbalance dataset can also be ignored while data pre-processing. The training verbose information introduced in Figure 16 shows the proper running of LGBM without getting in too much detail it can be simply noticed that the model did not converge to infinity.

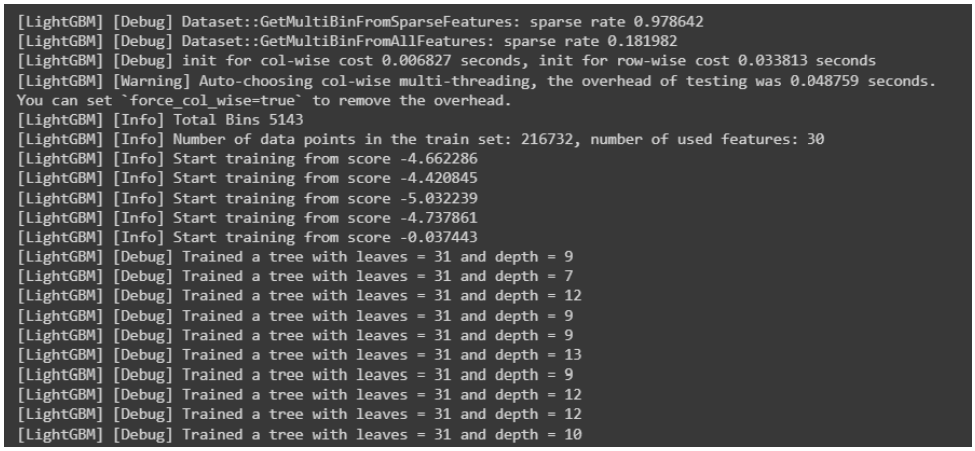


Figure 16: LGBM Verbose (Method 2)

Feature Importance

Figure 17: Feature Importance Gradient Boosting (Method 2)

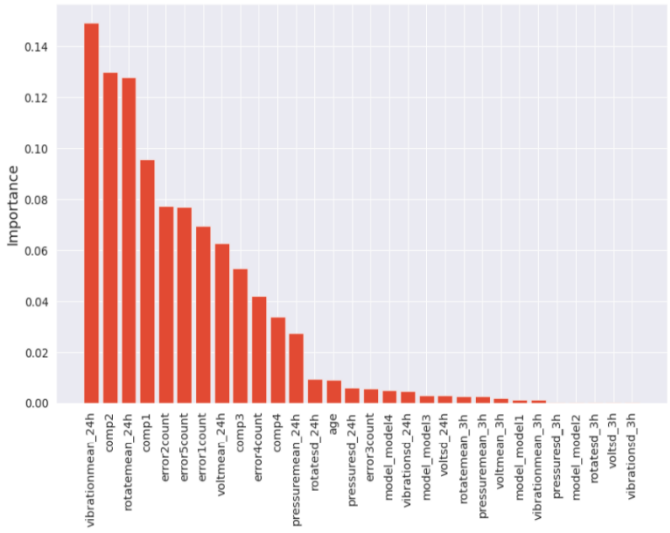
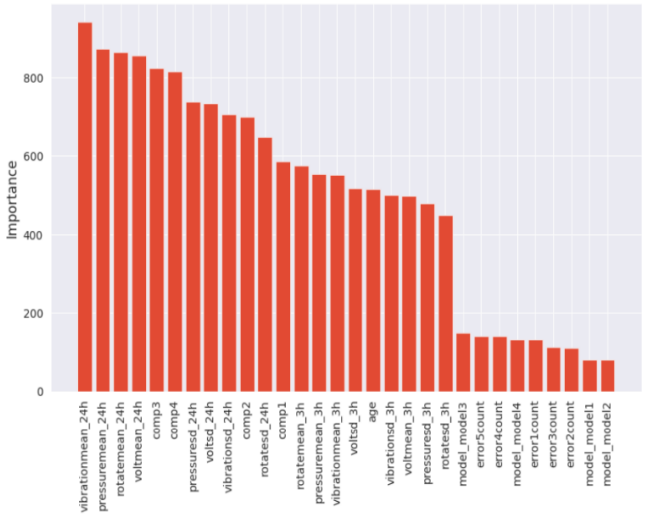


Figure 18: Feature Importance Light Gradient boosting (Method 2)



Figures 17 and 18 are called using the in-built specialized features for the gradient boosting type models. Both GBM and LGBM although learns from various weak models to create a strong one, have some differences that can lead to different features being considered important. LGBM uses an approach of binning the data and viewing it as histogram plots to find appropriate condition split. to find the best split for each feature, while GBM uses exact greedy algorithm. This can lead to different splits and consequently different feature importance. Regularization techniques in LGBM includes L1 (Lasso) and L2 (Ridge) regularization, which can also lead to a different model and consequently jumbled feature being considered important.

Randomness: Both algorithms include some randomness (for example, in the sampling of the data). If the datasets or initial conditions are slightly different, the resulting models and important features may also differ. Adding on to this, LGBM uses leaf-wise (best-first) tree growth, while traditional Gradient Boosting uses depth-wise tree growth. This can lead to trees of different shapes and consequently different feature importance.
It's important to understand that feature importance can vary depending on the model and the data. It's always a good idea to use different models and methods for feature selection and see if there is a consensus on the most important features.

Since the attributes are correlated in nature with the failure's records, Feature Importance plot does not show what was already known. However, it still confirms that simple moving averages on 24 hours basis specifically vibration of 24 hours encompass over 14% of the feature importance. Other sensors readings, lags from maintenance records, lags from error records, age and 3 hours data are also influential factors.

Understanding the Causality versus the correlation in important to not over-interpret the results here. Suppose if maintenance is a causal factor that directly affects failure, including it in the model is valid and necessary.

Classification report

The function evaluates computes various classification evaluation metrics for the given set of actual and predicted labels. The metrics computed by the function include:
The function computes and prints the confusion matrix, then computes the regular (per-class) precision, recall, F1 score, the macro-micro averaged versions of these metrics, then computes the average accuracy, the majority class metrics, and finally computes the expected accuracy and kappa statistic. The function returns all these metrics in a single pandas Data Frame. The result for GBM is mentioned in Table 5. The runtime on T4 GPU of the 3 different train and test sets totalled approximately 44 mins to train and predict.

Table 5: Performance results GBM

	none	comp1	comp2	comp3	comp4
accuracy	0.996795	0.996795	0.996795	0.996795	0.996795
precision	0.998816	0.874624	0.956113	0.954667	0.976424
recall	0.998267	0.926674	0.959119	0.961074	0.975466
F1	0.998542	0.899897	0.957614	0.957860	0.975945
macro precision	0.952129	0.952129	0.952129	0.952129	0.952129
macro recall	0.964120	0.964120	0.964120	0.964120	0.964120
macro F1	0.957971	0.957971	0.957971	0.957971	0.957971
average accuracy	0.998718	0.998718	0.998718	0.998718	0.998718
micro-averaged precision/recall/F1	0.996795	0.996795	0.996795	0.996795	0.996795
majority class accuracy	0.964973	0.000000	0.000000	0.000000	0.000000
majority class recall	1.000000	0.000000	0.000000	0.000000	0.000000
majority class precision	0.964973	0.000000	0.000000	0.000000	0.000000
majority class F1	0.982174	0.000000	0.000000	0.000000	0.000000
expected accuracy	0.930998	0.930998	0.930998	0.930998	0.930998
kappa	0.953551	0.953551	0.953551	0.953551	0.953551

In Contrast, LGBM results are shown in Table 6. While the results are almost equal to the GBM model seen above, the training and predicting time for the 3 different sets created totalled 17 seconds on T4 GPU. Thus, deciding the best fit model really depends on either the performance based on Kappa values and total runtime for effective studies.

	none	comp1	comp2	comp3	comp4
accuracy	0.996328	0.996328	0.996328	0.996328	0.996328
precision	0.997840	0.890877	0.968096	0.987500	0.966252
recall	0.998582	0.892473	0.934552	0.954106	0.959436
F1	0.998211	0.891674	0.951028	0.970516	0.962832
macro precision	0.962113	0.962113	0.962113	0.962113	0.962113
macro recall	0.947830	0.947830	0.947830	0.947830	0.947830
macro F1	0.954852	0.954852	0.954852	0.954852	0.954852
average accuracy	0.998531	0.998531	0.998531	0.998531	0.998531
micro-averaged precision/recall/F1	0.996328	0.996328	0.996328	0.996328	0.996328
majority class accuracy	0.965072	0.000000	0.000000	0.000000	0.000000
majority class recall	1.000000	0.000000	0.000000	0.000000	0.000000
majority class precision	0.965072	0.000000	0.000000	0.000000	0.000000
majority class F1	0.982226	0.000000	0.000000	0.000000	0.000000
expected accuracy	0.932395	0.932395	0.932395	0.932395	0.932395
kappa	0.945690	0.945690	0.945690	0.945690	0.945690

Table 6: Performance Evaluation LGBM (Method 2)

Figure 19: Confusion Matrix Gradient Boosting

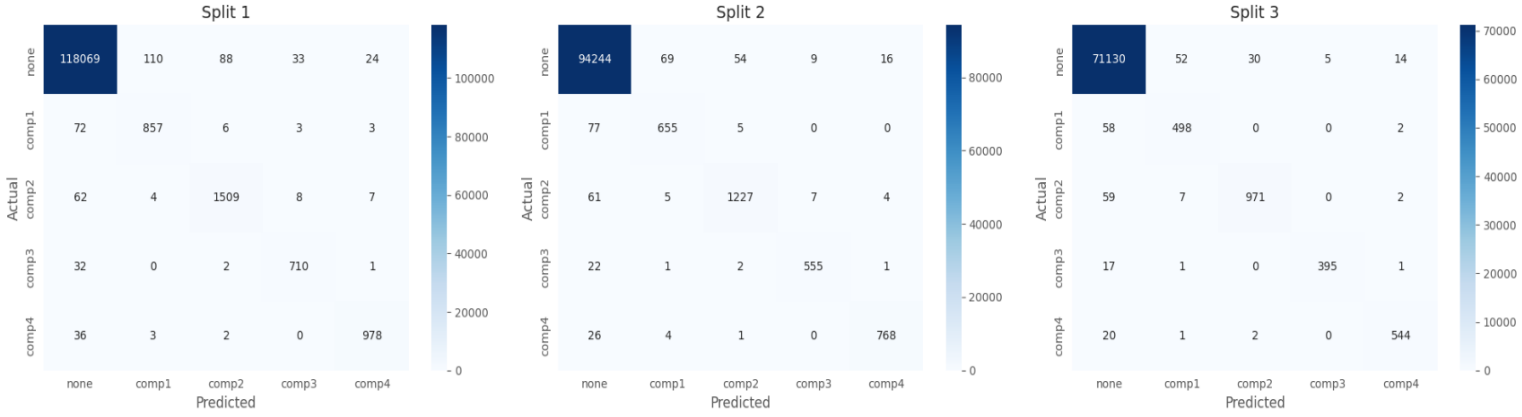


Figure 20: Confusion Matrix Light Gradient Boosting

The confusion matrix in Figure 19 and Figure 20 given above illustrates the performance of method 2 on both the ensemble-based algorithms for which the accuracy scores were calculated. Note 3 different splits on different time durations is expected to prepare a robust model for real-world use. The values on the diagonal are the true values and correctly predicted values. While the Majority Class of no failure was simple to predict for any model even when implementing method 1 earlier, the main concern remains in checking how well is the model able to distinguish between types of failures.

Knowing which type of failure is important when bringing the model into practice. By this, technicians are supposed to benefit with maintenance processes when they are able to understand where the problem within the EOT Crane remains to avoid dismantling the whole equipment to rectify the root cause.

An added feature of gradient boosting type models is the default function to call the best fit conditioning splits on which the models are based. Figure 21 shows an example of one such tree obtained after training the LGBM model.

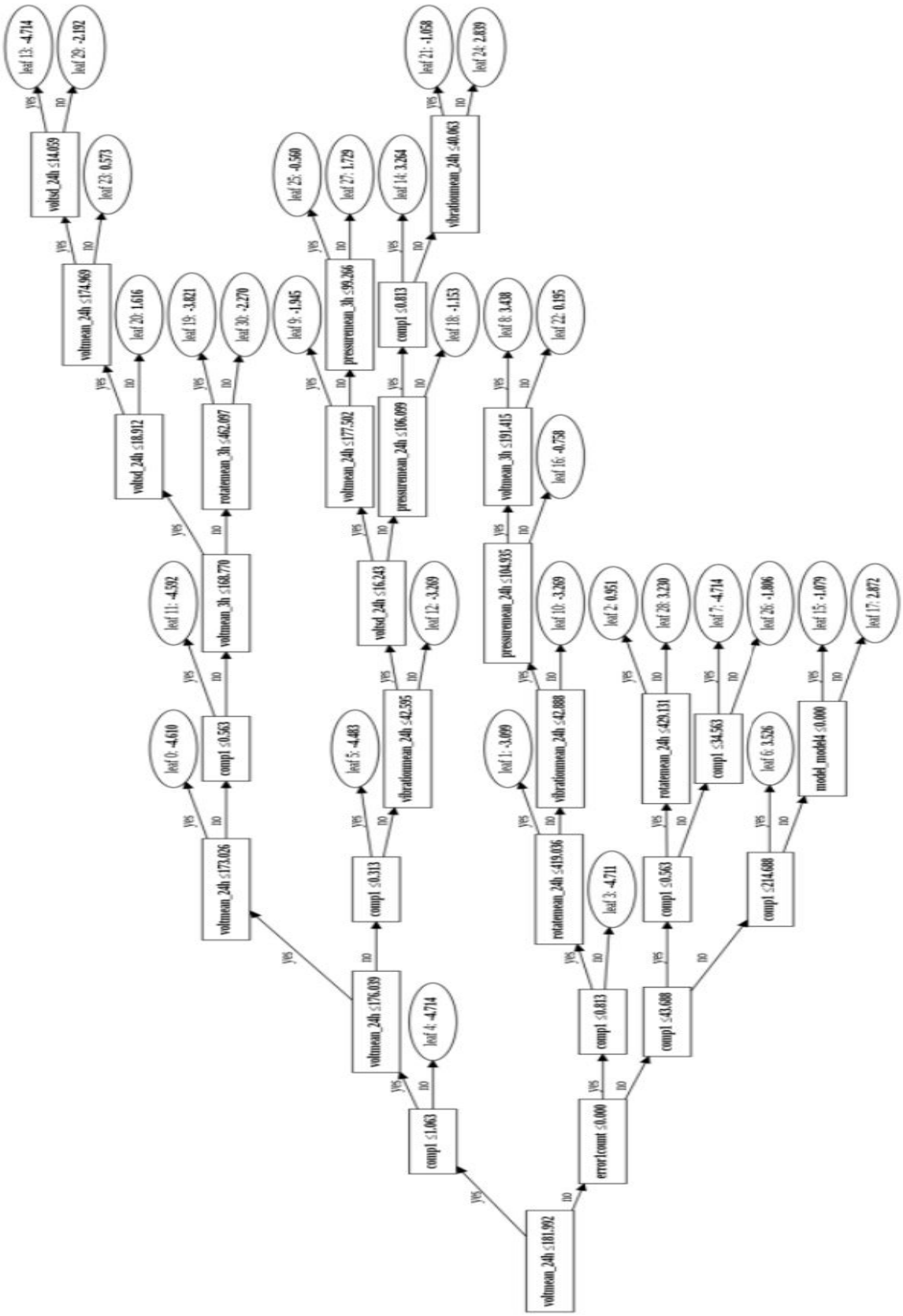


Figure 21: Most optimal Decision Tree (condition splitting) for First Train-Test Split acquired from LGBM's in-built function. [method 2 best fit model best condition split]

6. Conclusion

6.1 Summary and Recommendations

To conclude, Fluctuations in frequencies of the sensors can affect a machine's performance and overall life. A very high or low reading could be the cause of unalignment in the machine, but this does not result in the failure. To determine the failure, a component level study becomes essential. Interestingly, the signs of maintenance can be seen when the readings deviate completely from the normal state in its overall performance over time. Information about maintenance and errors are also the key factors affecting the failure of the machinery and these features must be smartly engineered to obtain high scores. Simultaneously, chances of experiencing a failure are one in a thousand, therefore failures records for multiple similar kinds of machines improves the quality of data.

Cloud platforms like Google Collaboratory are efficient platforms to carry out timely analysis and allow the user with all basic requirements needed during the analytics stage. Survival Analysis tests like Kaplan Meier Fitter can help formulate the schedule for preventive maintenance as seen that component 1 in 50% of machines be it of any model type is bound to fail in approximately 380 hours of its replacement. Similarly, average life for each of the components could help create an optimal schedule for logistics.

Testing the model on rolling window validation can guarantee that the model generalizes well to unseen data with seasonal patterns since the underlying patterns may change over time. Analysis of Historical data is a cost-effective and non-obligatory method for solving the real-world problem. While Gradient Boosting is proven to be highly accurate during the testing phase, but the operations are relatively slower (44 mins on T4 GPU) as compared to LGBM that presents almost equal results but with quick training time (17 seconds on T4 GPU).

minimum FP rate, maximizing precision rate and Kappa static should be the key evaluation parameters when choosing the best fit model if a few hours of runtime can be spared. Ensemble-based techniques prove effective in clustering while predicting while Fourier transformations is essential in time-series analysis for informed decision making. Looking for autocorrelation in the time-series can provide important information about past how many values could affect the model's performance as well.

6.2 Further work

The never-ending task of data analysis must be shifted to data lake platforms for acquiring the raw data, efficiently prepared and top of the line ML algorithms could be deployed with the same architecture. This will allow integrity in the operations and predictions for other stuff such a optimal delivery route etc could also be developed. To top it off, Alteryx is a talked about

environment to carry out the project which could be explored, this may allow one to create more informative flow charts and support other research related work when presenting the findings.

Multi-step time series forecasting, to make predictions for multiple future time steps beyond the current time point often requires modelling the dependencies between future time steps, which can be more complex than single-step forecasting (where you predict just the next time step). Techniques commonly used for multi-step time series forecasting include Vector Autoregression (VAR), Long Short-Term Memory (LSTM) networks, and other deep learning models designed for sequence-to-sequence prediction. These techniques have also made a mark in the market and could be explored.

When considering Single-step forecasting, direct lags and recursive lags could both be taken as features in the training set. Nevertheless, Simple moving average based on Fourier transformed also demands more research. Finally, binning the data just how LGBM bins the data during training can also lead to faster training time without. Finally, validation with real data was not possible though it is believed that the results show the potential to be used for real world application. In future, a cross-validation could also perhaps be conducted to test the quality of data and the performance of models.

7. Self-Evaluation

Firstly, I would like to appreciate the generic nature of this problem. I found this dataset remarkably interesting and engaging. Exploring the intricacies in dealing with time series proved challenging yet made me increasingly curious. The progress felt slow at the beginning since multiple papers had to be reviewed, strenuous feature engineering and data preparation techniques were to be followed, slow training time of many machine learning algorithms and hyperparameter tuning also seemed time consuming. I could not feel anything but overwhelmed at the time although systematically going through the literature review, revising time-series concepts, and dealing with each column separately turned out to be a better approach. Regular meetings with the supervisor and feedback from peer review helped me shape the report better. The Gantt Chart in Figure 22 shows the intended plan decided at the beginning of the research.

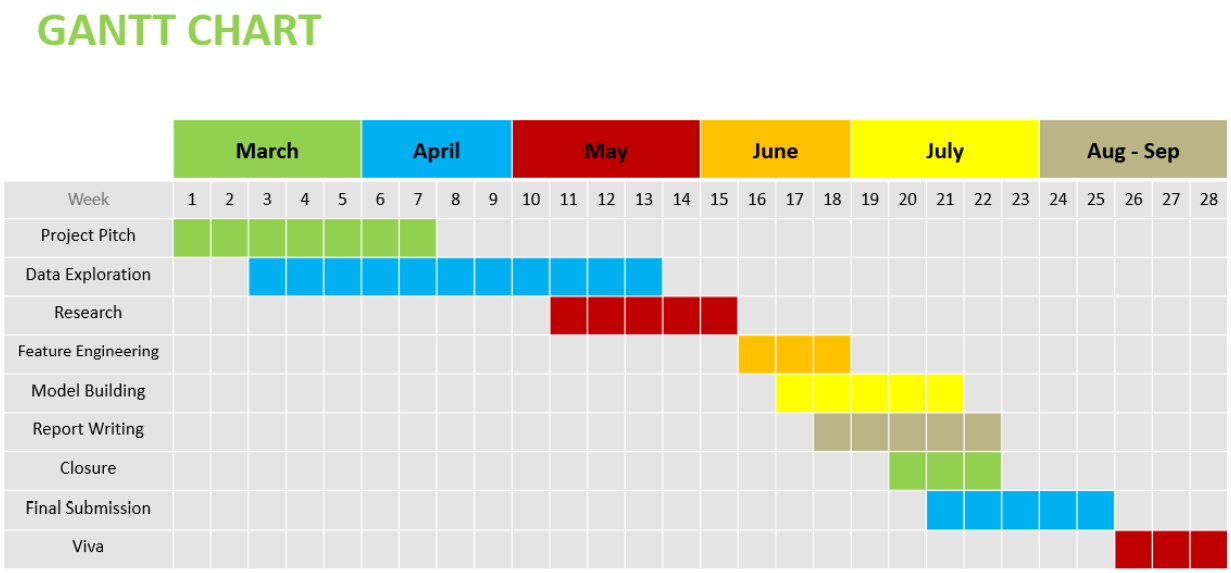


Figure 22: Gantt Chart as Planned in beginning of the project.

Although, majority of the timeline was based around similar dates, Literature review was not included here and should have been the first thing in before getting into data exploration. In addition, it was also believed that only deep-learning technique like LSTM would seem the optimal solution here but upon going through the literature review, it was found that ensemble-based approaches could also be as good as neural networks. The time provided was sufficient and the quality of data were other important factors that enriched this study.

8. References

1. KoneCranes (2019), *'Ten Ways to improve overhead Crane Safety'*, Finland
2. Andreassen O.R. (2018), *'Reliability centred maintenance of Electric Overhead Traveling (EOT) cranes installed in the well intervention area: A case study for ConocoPhillips'*, MSc Thesis, Universitetet i Stavanger, Norway
3. B. Feng, G. Pedrielli, Y. Peng, S. Shashaani, E. Song, C.G. Corlu, L.H. Lee, E.P. Chew, T. Roeder, and P. Lendermann, eds (2022), *'Predictive Maintenance Powered by Machine Learning and Simulation'*, 2022 Winter Simulation Conference (WSC), Available at: DOI:10.1109/WSC57314.2022.10015520 [Last accessed 01-09-23]
4. Hadi, S., Gustopo, D., Indra, D. (2019), *'Predictive Maintenance Analysis Overhead Crane Machine in PT Bromo Steel Indonesia'*, International Conference on Science and Technology 2019, IOP Publishing, 1569 (2020) 0022093, Available at: doi:10.1088/1742-6596/1569/2/022093 [Last Accessed 01.09.23]
5. Panagou, S., et al. (2022), *'Feature Investigation with Digital Twin for Predictive Maintenance following a Machine Learning Approach'*, IFAC PapersOnLine 55-2, pp. 132–137
6. Förderer K.H., Anderson T. and Zimmermann B., (2015), *'Predictive Maintenance for Automated and Manual Overhead Bridge Cranes',* Association for Iron & Steel Technology, Available at: https://www.psi-technics.com/PDF/Presse/E/PSI-Technics-Iron-and-Steel-Technology-News-Reprint-Predictive-Maintenance-for-Automated-and-Manual-Overhead-Cranes-June-2015.pdf
7. Flovik, V. (2019), *'Machine Learning for Anomaly Detection and Condition Monitoring',* blog post, Towards Data Science, 23 April, Available at: https://towardsdatascience.com/machine-learning-for-anomaly-detection-and-condition-monitoring-d4614e7de770 [Last Accessed 01.09.23]
8. Nanekar, O., (2022), *'Automatically Providing Predictive and Preventative Maintenance of Cranes used at Jaguar Land Rover car manufacturer via Machine Learning/AI',* MSc Thesis, Livepool John Moores University, Liverpool
9. Ameritrade, TD. (2021), *'How to use Moving Average for Stock Trading'*, YouTube, Available at: https://www.youtube.com/watch?v=r3Ulu0jZCJI&ab_channel=TDAmeritrade [Last Accessed 01.09.23]
10. Nabriya, P. (2021), *'Feature Engineering on Time-Series Data for Human Activity Recognition'*, blog post, Towards Data Science, 29 June, Available at: https://towardsdatascience.com/feature-engineering-on-time-series-data-transforming-signal-data-of-a-smartphone-accelerometer-for-72cbe34b8a60 Last Accessed [01.09.23]
11. Reducible (2021), *'The Fast Fourier Transform: Most Ingenious Algorithm Ever?'*, YouTube, Available at:https://www.youtube.com/watch?v=h7apO7q16V0&list=PLpUABdF8qfb6o8vawjMSBUF2yvgZGEwwa&index=6&ab_channel=Reducible [Last accessed 2021]

12. PyData (2022), *'Kishan Manani - Feature Engineering for Time-Series Forecasting | PyData London 2022'*, YouTube, Available at: https://www.youtube.com/watch?v=9QtL7m3YS9I, Last Accessed [01.09.23]
13. Javatpoint (2019), *'Logistical Regression in Machine Learning'*, blog post, Javatpoint, Available at: https://www.javatpoint.com/logistic-regression-in-machine-learning Last Accessed [01.09.23]
14. Normalized Nerd (2021), *'Decision Tree Classification Clearly Explained!'*, YouTube, Available at: https://www.youtube.com/watch?v=ZVR2Way4nwQ&list=PLpUABdF8qfb7Dmnn5Y93DNCViZKGw1vym&index=7&t=535s&ab_channel=NormalizedNerd [Last accessed 01.09.23]
15. Azami, R.; Lei, Z., Hermann, U., Zubick, T. (2022), *'A Predictive Analytics Framework for Mobile Crane Configuration Selection in Heavy Industrial Construction Projects'*, Buildings 2022, 12, 960, Available at: https://doi.org/10.3390/buildings12070960 Last Accessed [01.09.23]
16. Physiotutors (2017), *'Kappa Value Calculation | Reliability'*, YouTube, Available at: https://www.youtube.com/watch?v=DfNo32nL_fo&list=PLpUABdF8qfb7Dmnn5Y93DNCViZKGw1vym&index=8&ab_channel=Physiotutors [Last accessed 1-09-23]
17. Institute of Quality and Reliability (2019), *'Life Data Analysis of Complete data using Minitab Software'*, YouTube, Available at: https://www.youtube.com/watch?v=JuQ_AEtj1vE&list=PLpUABdF8qfb6tnU0drfOndqJP14jK8lJ6&index=14&t=1s&ab_channel=InstituteofQualityandReliability [Last accessed 01-09-23]
18. Wikipedia Encyclopaedia, Moving Average, Available at: https://en.wikipedia.org/wiki/Moving_average [Last accessed: 01.09.23]

9. Appendix

9.1 Supplement A: Exploratory Data Analysis Using Visual Studio Code

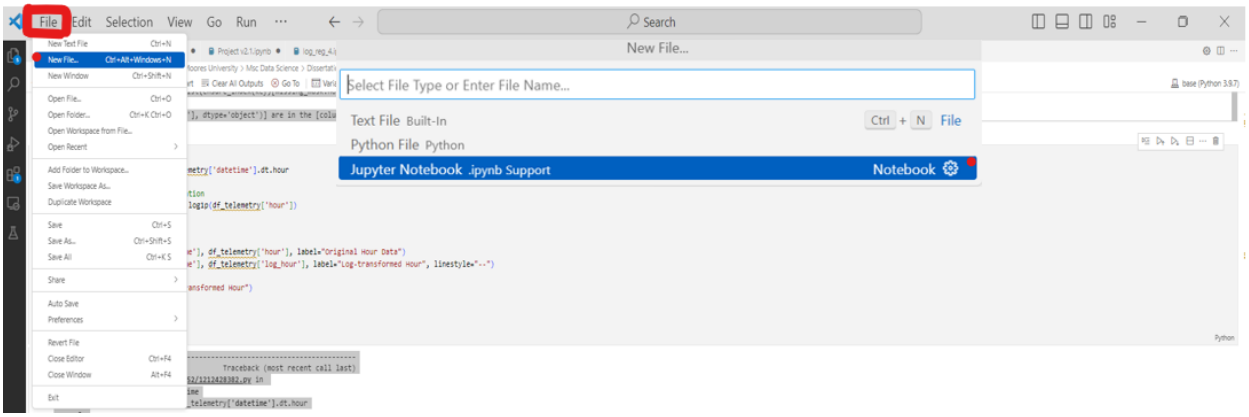


Illustration 0-1: Setting Jupyter Notebook Using Visual Studio Code

```
import time
#measure the start time - will be used to test the model's operation time
start = time.time()
print(start)
```
[54] ✓ 0.0s

```
1694053492.8437307
```

```
import pandas as pd
from sympy import * #Enables latex printing and creating equations for explanations
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math #math.pi to get exact pi value
import warnings
warnings.filterwarnings('ignore')

init_printing()
```
[55] ✓ 0.0s

```
import multiprocessing
multiprocessing.cpu_count() # Use Parallelization to save time in outgut generation
```
[56] ✓ 0.2s

```
8
```

```
errors = pd.read_csv("PdM_errors.csv")
failures = pd.read_csv("PdM_failures.csv")
machines = pd.read_csv("PdM_machines.csv")
telemetry = pd.read_csv("PdM_telemetry.csv")
maint = pd.read_csv('PdM_maint.csv')
```
[57] ✓ 1.3s

```
end = time.time()
print(end)
elapsed_time = end - start
print("Elapsed time:" , elapsed_time , "Seconds")
```
[58] ✓ 0.0s

```
1694053498.026388
Elapsed time: 5.182657241821289 Seconds
```

Utility Code 1: Demonstration of time library to get a stopwatch in python (May help in ML development)

 # Data Preparation and Exploratory Analysis

Importing the libraries and reading the files

```
import sys #Get version information
print(sys.version)

#Import necessary libraries for Data Exploration
import pandas as pd
import numpy as np

# Setting the Aesthetics
import seaborn as sns
import matplotlib
import matplotlib.pyplot as plt
matplotlib.style.use("Solarize_Light2")
%matplotlib inline
color_pal = sns.color_palette()

import warnings
warnings.filterwarnings('ignore')
```
✓ 26.1s

```
3.9.7 (default, Sep 16 2021, 16:59:28) [MSC v.1916 64 bit (AMD64)]
```

```
df_failure = pd.read_csv('PdM_failures.csv')
df_errors = pd.read_csv('PdM_errors.csv')
df_machines = pd.read_csv('PdM_machines.csv')
df_maint = pd.read_csv('PdM_maint.csv')
df_telemetry = pd.read_csv('PdM_telemetry.csv')
```
✓ 1.0s

Creating default functions for exploration and data transformation

```
def get_data_info(df):
    print("-"*50)
    print("Number of rows: ",df.shape[0])
    print("Number of Columns: ",df.shape[1])
    print("-"*50)
    print("Column-wise Missing values: ")
    print(df.isna().sum())
    print("-"*50)
    print("Count of mentioned data types")
    print(df.dtypes.value_counts())
    print("-"*50)
    print("Duplicate instances:", df.duplicated().sum())


def get_float_type_columns(df):
    return df.columns[df.dtypes == "float64"]
```

```
✓ 0.0s
```

```
# Box plot with groupby
def box_with_groupy (df, attribute, by):
    """
    attribute_name: Name of the feature to be plotted
    by: Name of the feature based on which groups are created [monthly in this case]
    """
    df.boxplot(column=col_name, by=by, vert=False,
                           figsize=(10, 6))
    plt.title(f"Distribution of {col_name} by {by}")
    plt.show()

#Plot barh for a particular feature(kind : Type of the plot, color : Color of the bars, check normalize and add percentage for better info.
def plot_barh(df, feature_name, normalize=True,
              kind='barh', figsize=(15, 5), sort_index=False, title=None, color=color_pal):

    counts = df[feature_name].value_counts(normalize=normalize, dropna=False)

    if sort_index:
        counts = counts.sort_index()
    else:
        counts = counts.sort_values()

    ax = counts.plot(kind=kind, figsize=figsize, grid=True, color=color_pal, title=title)

    # Add numbers on top of the bars as percentages
    for i, v in enumerate(counts):
        percentage_value = f"{v*100:.2f}%" if normalize else f"{v:.2f}"
        ax.text(v, i, percentage_value, color='grey', va='center')

    plt.xlabel('Count' if not normalize else 'Percentage')
    plt.ylabel(feature_name)
    plt.show()
```

```
def merger_with_duplicate_row_remover(df1 , df2 ):
    print("*"*200)
    if ("datetime" in df2.columns):
        merged_df =pd.merge(df1, df2, on=['datetime','machineID'],how='left')
        merged_df = merged_df.replace(np.NaN,0)
        print("Shape of left dataset:                          ",df1.shape)
        print("Shape of the right dataset:                     ",df2.shape)
        print("Shape of merged dataset before checking duplicates:",merged_df.shape)

        #creating an extra column that will have unique datetime+machineID
        merged_df['combo'] = merged_df['machineID'].astype(str) + merged_df['datetime'].astype(str)
        # merged_df['combo'].value_counts() to check duplicates Anything greater than 1 will be duplicated
        li = merged_df['combo'].value_counts()
        valids = li[li > 1].index
        print("Duplicate rows found:", len(valids))

        merged_df[merged_df['combo'].isin(valids)] #create a dataframe To get rows of deficit indices
        # Here dropping the duplicate rows becomes essential
        merged_df = merged_df.drop_duplicates(subset=['combo'])
        print("Duplicates rows removed:", len(valids)/2 )
        print("Shape of merged dataset after removing duplicate columns:", merged_df.shape)
    else:
        # Machine dataframe has no datatime plus no duplicates
        merged_df =pd.merge(df1, df2, on=['machineID'],how='left')
        merged_df = merged_df.replace(np.NaN,0)
        print("Shape of left dataset:                          ",df1.shape)
        print("Shape of the right dataset:                     ",df2.shape)
        print("Shape of merged dataset before checking duplicates:",merged_df.shape)

    return merged_df
```

```
df = merger_with_duplicate_row_remover(df_telemetry, df_failure)
df = merger_with_duplicate_row_remover(df1 = df, df2 = df_errors )
df = merger_with_duplicate_row_remover(df1 = df, df2 = df_maint )
df = merger_with_duplicate_row_remover(df1=df,df2=df_machines)
df['datetime'] = pd.to_datetime(df['datetime'])
df['date'] = df['datetime'].dt.date
df['hour'] = df['datetime'].dt.strftime('%H:%M:%S')
```

Errors

```
[7]
df_errors['errorID'].unique()
```

```
array(['error1', 'error3', 'error5', 'error4', 'error2'], dtype=object)
```

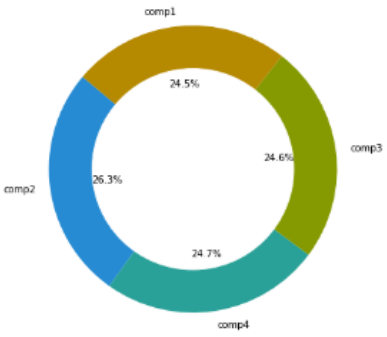
```
[8]
import squarify #pip install squarify

# Calculate value_counts of errorID
error_counts = df_errors['errorID'].value_counts()

# Create a data frame from the value_counts
df_error_counts = pd.DataFrame({'count': error_counts.values, 'errorID': error_counts.index})

# plot it using squarify
plt.figure(figsize=(8, 6))  # Adjust the figure size if needed
squarify.plot(sizes=df_error_counts['count'], label=df_error_counts['errorID'], alpha=.8 )
plt.axis('off')
plt.title('Error type counts')
plt.show()
```

Maintenance Dataset

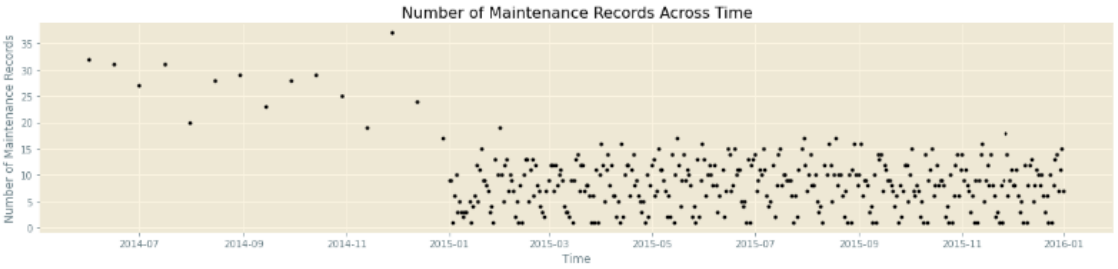
```
[9]
df_maint.keys()
```

```
Index(['datetime', 'machineID', 'comp'], dtype='object')
```

```
# Donut Chart # Calculate value_counts of comp
comp_counts = df_maint['comp'].value_counts()

# Create a data frame from the value_counts
df_comp_counts = pd.DataFrame({'count': comp_counts.values}, index=comp_counts.index)

# Create a pie plot
plt.figure(figsize=(6, 6))  # Adjust the figure size if needed
plt.pie(df_comp_counts['count'], labels=df_comp_counts.index, autopct='%1.1f%%', startangle=140)
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

# add a circle at the center to transform it into a donut chart
my_circle = plt.Circle((0, 0), 0.7, color='white')
p = plt.gcf()
p.gca().add_artist(my_circle)
plt.show()
```




```
df_maint['datetime'] = pd.to_datetime(df_maint['datetime'])
# Extract date from datetime
df_maint['date'] = df_maint['datetime'].dt.date

df_maint['date'].value_counts().plot(
    style="k.",
    figsize=(20, 4),
    title="Number of Maintenance Records Across Time"
)
plt.ylabel("Number of Maintenance Records")
plt.xlabel("Time")
plt.show()
```

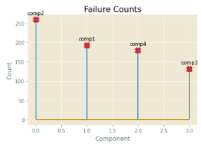


Failures Data

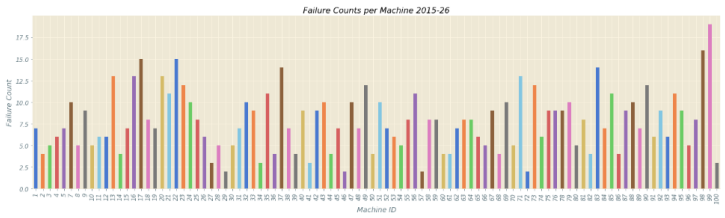
```
get_data_info(df_failure)
print("Total number of machines that recorded failure at any point:" , len(df_failure['machineID'].value_counts()) )
print("Maximum failures recorded by any machine ID:" , df_failure['machineID'].value_counts().max())
print("Minimum frquency of failures by one of the machineID:" , df_failure['machineID'].value_counts().min())
not_found = []
for i in range(1,101):
    if i in df_failure.machineID.unique():
        pass
    else:
        not_found.append(i)
print("MachineIDs with no failures records:", not_found[0],"&", not_found[1])
```

```
--------------------------------------------------
Number of rows:  761
Number of Columns:  3
--------------------------------------------------
Column-Wise missing values:
datetime     0
machineID    0
failure      0
dtype: int64
--------------------------------------------------
Count of mentioned data types
object    2
int64     1
dtype: int64
--------------------------------------------------
Duplicate instances: 0
Total number of machines that recorded failure at any point: 98
Maximum failures recorded by any machine ID: 19
Minimum frquency of failures by one of the machineID: 2
MachineIDs with no failures records: 6 & 77
```

```
# Create a DataFrame to represent the value counts
value_counts = df_failure['failure'].value_counts()

# Get the values and labels from the value counts
values = value_counts.values
labels = value_counts.index

# Create a stem plot
(markers, stemlines, baseline) = plt.stem(values)
plt.setp(markers, marker='X', markersize=10, markeredgecolor="red", markeredgewidth=2)

# Add labels to the markers
for i, label in enumerate(labels):
    plt.annotate(label, (i, values[i]), textcoords="offset points", xytext=(0, 10), ha='center')

# Customize the plot
plt.title('Failure Counts')
plt.xlabel('Component')
plt.ylabel('Count')

# Show the plot
plt.show()
```




```
from sklearn import preprocessing
df2 = df_failure.copy()
label_encoder = preprocessing.LabelEncoder()

# Encode labels in column 'species'.
df2['failure_encoded'] = label_encoder.fit_transform(df2['failure'])
df2['failure_encoded'] += 1
df2['failure'].unique()
avg_failure = df2.groupby('machineID')['failure_encoded'].size()


# Create the bar plot
plt.figure(figsize=(20, 6))
avg_failure.plot(kind='bar', color=color_pal)

# Customize the plot
plt.title('Failure Counts per Machine 2015-26', fontsize=16, fontstyle='italic')
plt.xlabel('Machine ID', fontsize=14, fontstyle='italic')
plt.ylabel('Failure Count', fontsize=14, fontstyle='italic')
plt.xticks(rotation=90, fontsize=12, fontstyle='italic')
plt.yticks(fontsize=12, fontstyle='italic')
sns.despine()  # Remove top and right spines

# Show the plot
plt.tight_layout()
plt.show()
```



Telemetry

```
display(df_telemetry.tail(1))
```

	datetime	machineID	volt	rotate	pressure	vibration
876099	2016-01-01 06:00:00	100	171.336037	496.096587	79.095538	37.845245

```
get_data_info(df_telemetry)
```

```
--------------------------------------------------
Number of rows:  876100
Number of Columns:  6
--------------------------------------------------
Column-Wise missing values:
datetime    0
machineID   0
volt        0
rotate      0
pressure    0
vibration   0
dtype: int64
--------------------------------------------------
Count of mentioned data types
float64   4
object    1
int64     1
dtype: int64
--------------------------------------------------
Duplicate instances: 0
```

```
sensors = ['volt', 'rotate', 'pressure', 'vibration']
df_pivot_telemetry = pd.pivot_table(df_telemetry, index='datetime', columns='machineID', values= sensors)
print(df_pivot_telemetry.shape)
display(df_pivot_telemetry)
```

```
(8761, 400)
```

	pressure										...	volt									
machineID	1	2	3	4	5	6	7	8	9	10	...	91	92	93	94	95	96	97	98	99	100
datetime																					
2015-01-01 06:00:00	113.077935	76.005332	87.453199	95.929877	84.591722	149.003582	105.878295	111.735733	102.874438	119.750673	...	177.688355	159.446409	154.356894	175.639054	201.977168	155.273542	143.613163	153.300953	168.596133	161.587466
2015-01-01 07:00:00	95.460525	110.907806	109.318603	92.691920	119.082137	146.822905	106.831651	109.004027	102.027505	88.860999	...	173.426162	189.422834	158.261613	166.829717	171.602257	147.249491	169.258107	171.471504	153.667693	163.333175
2015-01-01 08:00:00	75.237905	97.877007	101.285580	81.506614	90.212540	162.090315	96.961839	108.776671	93.187589	84.382394	...	157.582897	162.088177	167.533443	166.130186	178.198293	176.628231	162.714703	170.773931	178.319255	153.926056
2015-01-01 09:00:00	109.248561	84.440430	98.825669	96.265308	95.072898	133.647267	102.497877	97.887247	95.469670	105.870821	...	139.092689	174.951013	169.745685	169.926452	164.762482	183.274022	166.069217	165.851737	162.163821	174.653605
2015-01-01 10:00:00	111.886648	110.395683	102.800017	100.885645	76.874388	119.104654	105.237040	100.929357	104.072420	98.148214	...	156.584252	183.651525	171.956004	167.610602	163.891788	180.739105	167.869839	175.140637	180.201336	199.637542
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2016-01-01 02:00:00	85.197443	86.021068	112.680909	118.755521	153.187090	85.778629	95.675358	115.815266	97.591019	92.991267	...	170.314791	188.087304	202.858467	191.330100	162.023059	148.087457	168.756556	164.321319	166.719608	179.438162
2016-01-01 03:00:00	95.568265	101.744819	99.657113	93.214591	152.191926	87.103379	90.748555	98.935286	103.082124	80.534193	...	137.134341	155.680083	171.364527	191.313184	185.141953	161.739023	151.618281	180.410465	168.434193	189.617555
2016-01-01 04:00:00	95.483913	85.709946	100.083961	108.622013	142.451649	107.157858	101.410932	108.873397	88.031814	113.411559	...	171.885617	195.399561	182.867119	187.952789	167.419408	129.156343	194.995971	158.354201	191.505730	192.483414
2016-01-01 05:00:00	102.694358	93.547606	99.240117	83.377101	156.228631	86.786087	93.777156	96.516092	114.479362	91.922893	...	172.246494	175.033128	166.943545	180.425830	181.805247	185.392000	184.873489	193.754368	185.423810	165.475310
2016-01-01 06:00:00	100.893691	99.946293	91.721593	112.680499	142.414273	98.743578	107.488234	89.904141	93.243580	111.683489	...	165.002636	182.828969	186.324464	171.397961	189.171616	157.157424	159.607756	165.717790	168.439623	171.336037

8761 rows × 400 columns

```
get_data_info(df_pivot_telemetry)
```

```
--------------------------------------------------
Number of rows:  8761
Number of Columns:  400
--------------------------------------------------
Column-Wise missing values:
          machineID
pressure  1          0
          2          0
          3          0
          4          0
```

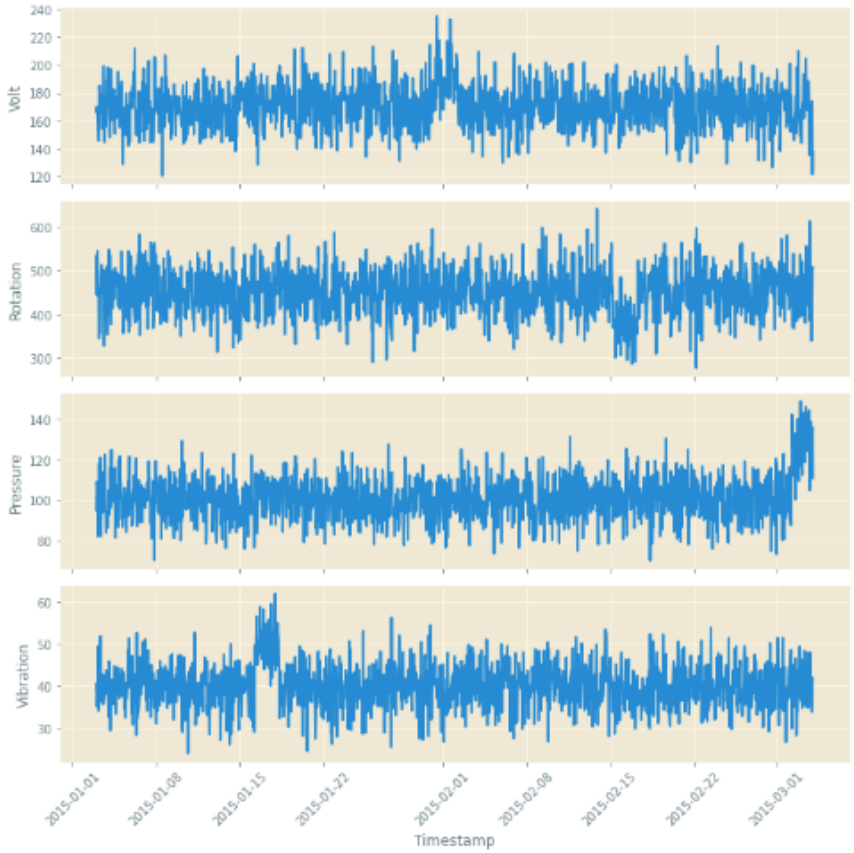
Since the shape is big, lets make a train set for data exploration.

```
# telemetry records of Machine ID: 99
mach_99 = df_telemetry.loc[df_telemetry['machineID'] == 99].reset_index(drop=True)

# Change datatype of the timestamp column from object to datetime
mach_99['datetime'] = pd.to_datetime(mach_99['datetime'])
```

now we can check these dates for machine 99 to check each failure component and sensors behaviour.

```
# Select the dates to check from failure records
# checking long term patterns by manually selecting the time period. 14 day interval in this case.
dates_to_check = ["2015-01-02", "2015-01-18", "2015-02-02", "2015-02-17"]

def plot_sensor_data(df, date, days_before=30, days_after=30, figsize=(10, 10), hspace=0.025):
    # Change datatype of the timestamp column from object to datetime
    df['datetime'] = pd.to_datetime(df['datetime'])

    # Find the index of the specified date
    st = df.loc[df['datetime'] == date].index.values[0]

    # Filter the telemetry data by the date and specified number of days before and after
    select = df.loc[st - days_before*24: st + days_after*24, :]

    # Create subplots
    fig, ax = plt.subplots(nrows=2, sharex=True, figsize=(10, 10))
    plt.subplots_adjust(hspace=hspace)
    # Plot sensor data
    ax[0].plot(select['datetime'], select['volt'])
    ax[0].set_ylabel("Volt")

    ax[1].plot(select['datetime'], select['rotate'])
    ax[1].set_ylabel("Rotation")

    ax[2].plot(select['datetime'], select['pressure'])
    ax[2].set_ylabel("Pressure")

    ax[3].plot(select['datetime'], select['vibration'])
    ax[3].set_ylabel("Vibration")

    ax[3].set_xlabel("Timestamp")
    ax[3].tick_params(axis='x', rotation=45)

    # Adjust space between subplots
    plt.tight_layout()
    plt.show()
```

```
plot_sensor_data(df = mach_99, date = "2015-02-02")
```




```
#Manually checking the fequency changes and comparing them with the failure records
print(df_failure[df_failure['machineID']== 99]['failure'][0:4])
print(df_failure[df_failure['machineID']== 99]['datetime'][0:4])
```

```
739    comp3
740    comp4
741    comp1
742    comp2
Name: failure, dtype: object
739    2015-01-02 03:00:00
740    2015-01-18 06:00:00
741    2015-02-02 06:00:00
742    2015-02-17 06:00:00
Name: datetime, dtype: object
```

1. failure in comp4 on 2015-01-18 show anomalous pattern in vibration.
2. failure in comp2 on 2015-02-17 show drastic decline in rotations per minute on that date.
3. failure in comp1 on 2015-02-02 was an effect of high voltage input recorded.
4. failure in comp3 can also be identified as must majorly relate to pressure.

9.2 Supplement B: Machine Learning using free-access T4 GPU (Google Collaboratory)

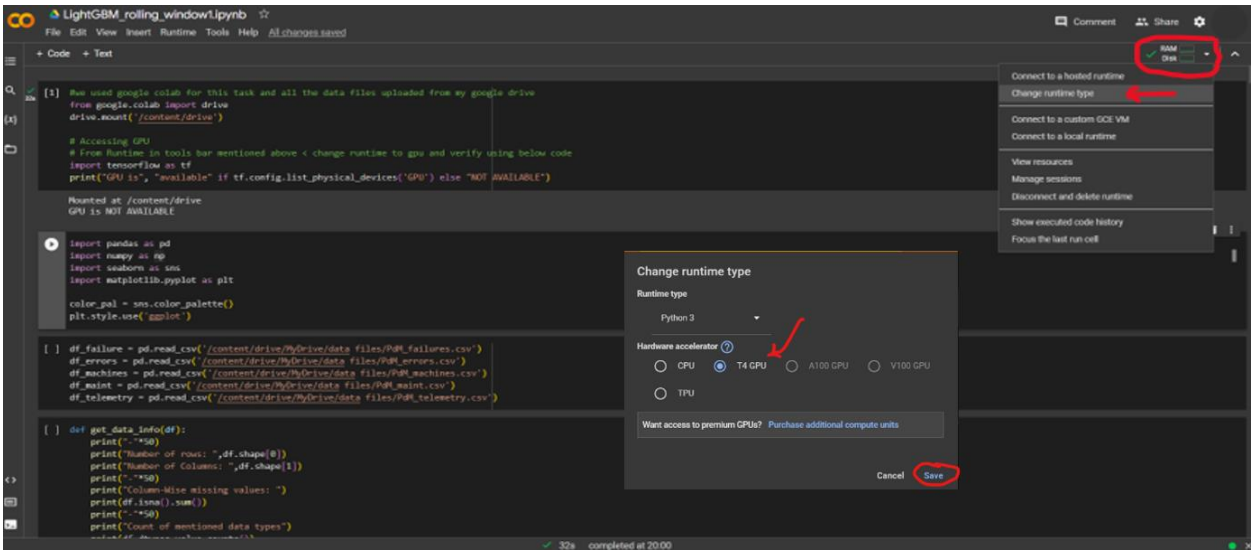


Illustration 0-2: Setting-up T4 GPU *# # A Google Account will be required to access Google Collaboratory website. Mounting Drive and reading the files using files using pathway*

```
[1]  #we used google colab for this task and all the data files uploaded from my google drive
     from google.colab import drive
     drive.mount('/content/drive')

     # Accessing GPU
     # From Runtime in tools bar mentioned above < change runtime to gpu and verify using below code
     import tensorflow as tf
     print("GPU is", "available" if tf.config.list_physical_devices('GPU') else "NOT AVAILABLE")

     Mounted at /content/drive
     GPU is available

[2]  import pandas as pd
     import numpy as np
     import seaborn as sns
     import matplotlib.pyplot as plt

     color_pal = sns.color_palette()
     plt.style.use('ggplot')

[3]  df_failure = pd.read_csv('/content/drive/MyDrive/data files/PdM_failures.csv')
     df_errors = pd.read_csv('/content/drive/MyDrive/data files/PdM_errors.csv')
     df_machines = pd.read_csv('/content/drive/MyDrive/data files/PdM_machines.csv')
     df_maint = pd.read_csv('/content/drive/MyDrive/data files/PdM_maint.csv')
     df_telemetry = pd.read_csv('/content/drive/MyDrive/data files/PdM_telemetry.csv')

[4]  def get_data_info(df):
         print("-"*50)
         print("Number of rows: ",df.shape[0])
         print("Number of Columns: ",df.shape[1])
         print("-"*50)
         print("Column-Wise missing values: ")
         print(df.isna().sum())
         print("-"*50)
         print("Count of mentioned data types")
         print(df.dtypes.value_counts())
         print("-"*50)
         print("Duplicate instances:", df.duplicated().sum())


     def get_float_type_columns(df):
         return df.columns[df.dtypes == "float64"]

[5]  # Failures
     df_failure['datetime'] = pd.to_datetime(df_failure['datetime'], format="%Y-%m-%d %H:%M:%S")
     df_failure['failure'] = df_failure['failure'].astype('category')
     # Errors
     df_errors['datetime'] = pd.to_datetime(df_errors['datetime'],format = '%Y-%m-%d %H:%M:%S')
     df_errors['errorID'] = df_errors['errorID'].astype('category')
     # Maintenance
     df_maint['datetime'] = pd.to_datetime(df_maint['datetime'], format='%Y-%m-%d %H:%M:%S')
     df_maint['comp'] = df_maint['comp'].astype('category')
     # Machines Meta
     df_machines['model'] = df_machines['model'].astype('category')
     # Telemetry
     df_telemetry['datetime'] = pd.to_datetime(df_telemetry['datetime'])
     df_telemetry.set_index('datetime', inplace=True)
```

```
# Calculate mean values for telemetry features
# https://www.youtube.com/watch?v=l4dvMiSDBzs # how resample works
# https://www.youtube.com/watch?v=KuT2n1w0Ixc&list=LL&index=1 ; explains pivot shift is good for reliability
temp = []
fields = ['volt', 'rotate', 'pressure', 'vibration']
for col in fields:
    temp.append(pd.pivot_table(df_telemetry,
                               index='datetime',
                               columns='machineID',
                               values=col).resample('3H', closed='left', label='right').mean().unstack())
telemetry_mean_3h = pd.concat(temp, axis=1)
telemetry_mean_3h.columns = [i + 'mean_3h' for i in fields]
telemetry_mean_3h.reset_index(inplace=True)

# repeat for standard deviation
temp = []
for col in fields:
    temp.append(pd.pivot_table(df_telemetry,
                               index='datetime',
                               columns='machineID',
                               values=col).resample('3H', closed='left', label='right').mean().unstack())
telemetry_sd_3h = pd.concat(temp, axis=1)
telemetry_sd_3h.columns = [i + 'sd_3h' for i in fields]
telemetry_sd_3h.reset_index(inplace=True)
telemetry_mean_3h.head(1)
```

	machineID	datetime	voltmean_3h	rotatemean_3h	pressuremean_3h	vibrationmean_3h
0	1	2015-01-01 09:00:00	170.028903	449.533798	94.592122	40.893502

```
[10] temp = []
fields = ['volt', 'rotate', 'pressure', 'vibration']
for col in fields:
    temp.append(df_telemetry.pivot_table(index='datetime', columns='machineID', values=col)
                    .rolling(window=24)
                    .mean()
                    .resample('3H', closed='left', label='right')
                    .first()
                    .unstack())
telemetry_mean_24h = pd.concat(temp, axis=1)
telemetry_mean_24h.columns = [i + 'mean_24h' for i in fields]
telemetry_mean_24h.reset_index(inplace=True)
telemetry_mean_24h = telemetry_mean_24h.loc[-telemetry_mean_24h['voltmean_24h'].isnull()]

# Calculate rolling standard deviations
temp = []
for col in fields:
    temp.append(pd.pivot_table(df_telemetry,
                               index='datetime',
                               columns='machineID',
                               values=col)
                .rolling(window=24)
                .std()
                .resample('3H', closed='left', label='right')
                .first()
                .unstack())
telemetry_sd_24h = pd.concat(temp, axis=1)
telemetry_sd_24h.columns = [i + 'sd_24h' for i in fields]
telemetry_sd_24h.reset_index(inplace=True)
telemetry_sd_24h = telemetry_sd_24h.loc[-telemetry_sd_24h['voltsd_24h'].isnull()]

# Notice that a 24h rolling average is not available at the earliest timepoints
telemetry_mean_24h.head(1)
```

```
# merge columns of feature sets created earlier
telemetry_feat = pd.concat([telemetry_mean_3h,
                            telemetry_sd_3h.iloc[:, 2:6],
                            telemetry_mean_24h.iloc[:, 2:6],
                            telemetry_sd_24h.iloc[:, 2:6]], axis=1).dropna()
telemetry_feat.keys()
```

```
Index(['machineID', 'datetime', 'voltmean_3h', 'rotatemean_3h',
       'pressuremean_3h', 'vibrationmean_3h', 'voltsd_3h', 'rotatesd_3h',
       'pressuresd_3h', 'vibrationsd_3h', 'voltmean_24h', 'rotatemean_24h',
       'pressuremean_24h', 'vibrationmean_24h', 'voltsd_24h', 'rotatesd_24h',
       'pressuresd_24h', 'vibrationsd_24h'],
      dtype='object')
```

Lags from Maintenance - get the difference of days for each component when it was last maintained

```
[ ] comp_rep = pd.get_dummies(df_maint.set_index('datetime')).reset_index()
    comp_rep.columns = ['datetime', 'machineID', 'comp1', 'comp2', 'comp3', 'comp4']

    # combine repairs for a given machine in a given hour
    comp_rep = comp_rep.groupby(['machineID', 'datetime']).sum().reset_index()
```

```
[ ] comp_rep = pd.get_dummies(df_maint.set_index('datetime')).reset_index() # create a column for each maintenance type
    comp_rep.columns = ['datetime', 'machineID', 'comp1', 'comp2', 'comp3', 'comp4']

    # combine repairs for a given machine in a given hour
    comp_rep = comp_rep.groupby(['machineID', 'datetime']).sum().reset_index()

    # add timepoints where no components were replaced
    df_telemetry.reset_index(inplace = True)
    comp_rep = df_telemetry[['datetime', 'machineID']].merge(comp_rep,
                                                             on=['datetime', 'machineID'],
                                                             how='outer').fillna(0).sort_values(by=['machineID', 'datetime'])

    components = ['comp1', 'comp2', 'comp3', 'comp4']
    for comp in components:
        # convert indicator to most recent date of component change
        comp_rep.loc[comp_rep[comp] < 1, comp] = None
        comp_rep.loc[-comp_rep[comp].isnull(), comp] = comp_rep.loc[-comp_rep[comp].isnull(), 'datetime']

        # forward-fill the most-recent date of component change
        comp_rep[comp] = comp_rep[comp].fillna(method='ffill')

    # remove dates in 2014 (may have NaN or future component change dates)
    comp_rep = comp_rep.loc[comp_rep['datetime'] > pd.to_datetime('2015-01-01')]

    # replace dates of most recent component change with days since most recent component change
    for comp in components:
        comp_rep[comp] = (comp_rep['datetime'] - comp_rep[comp]) / np.timedelta64(1, 'D')

    #comp_rep.describe()
```

Lags from errors

```
error_count = pd.get_dummies(df_errors.set_index('datetime')).reset_index()
error_count.columns = ['datetime', 'machineID', 'error1', 'error2', 'error3', 'error4', 'error5']
# combine errors for a given machine in a given hour
error_count = error_count.groupby(['machineID','datetime']).sum().reset_index()
error_count = df_telemetry[['datetime', 'machineID']].merge(error_count, on=['machineID', 'datetime'], how='left').fillna(0.0)
#display(error_count.describe())
# compute the total number of errors of each type over the last 24 hours, for timepoints taken every three hours
```

```
[18] temp = []
fields = ['error%d' % i for i in range(1, 6)] # creates strings like "error1"..."error5".

for col in fields:
    temp.append(
        pd.pivot_table(error_count,
                       index='datetime',
                       columns='machineID',
                       values=col)
        .rolling(window=24)
        .sum()
        .resample('3H', closed='left', label='right')
        .first()
        .unstack()
    )

error_count = pd.concat(temp, axis=1)
error_count.columns = [i + 'count' for i in fields]
error_count.reset_index(inplace=True)
error_count = error_count.dropna()
#error_count.describe()
```

```
[19] final_feat = telemetry_feat.merge(error_count, on=['datetime', 'machineID'], how='left')
final_feat = final_feat.merge(comp_rep, on=['datetime', 'machineID'], how='left')
final_feat = final_feat.merge(df_machines, on=['machineID'], how='left')
```

```
# Failure type 'none' couldn't be directly added to the column.
labeled_features = final_feat.merge(df_failure, on=['datetime', 'machineID'], how='left')
labeled_features = labeled_features.fillna(method='bfill', limit=7)
labeled_features = labeled_features.fillna(method='ffill', limit=7)
labeled_features['failure'] = labeled_features['failure'].cat.add_categories(['none'])

# Fill missing values with the 'missing' category
labeled_features['failure'].fillna('none', inplace=True)
```

+ Code + Text

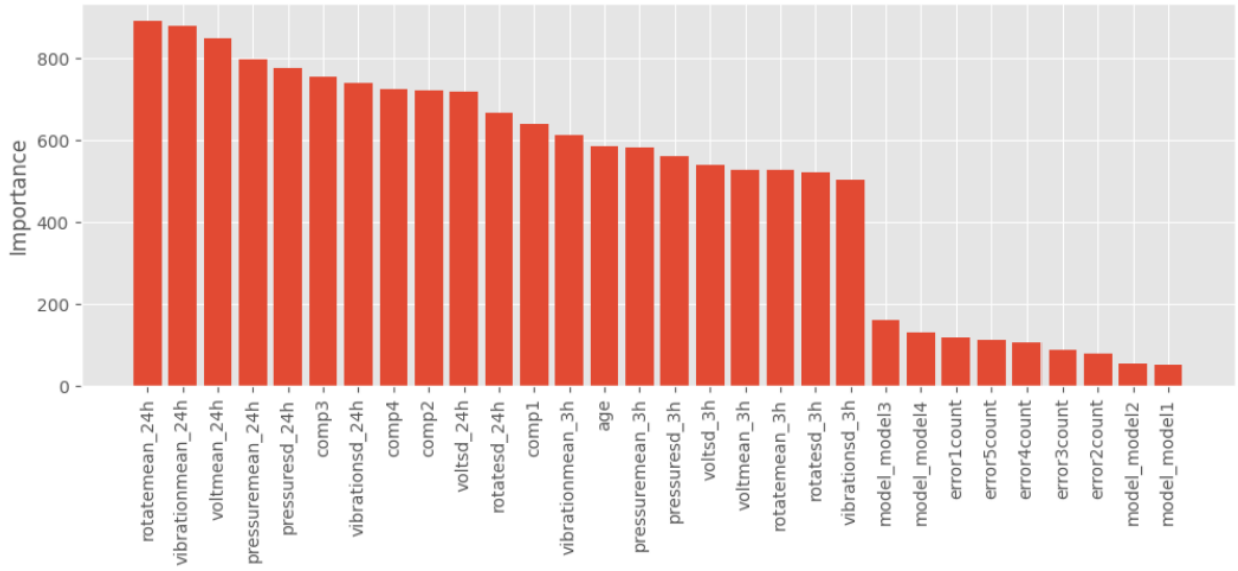
```
[21] #labeled_features.loc[labeled_features['failure'] == 'comp2'][:24]
```

```
# make test and training splits
# 3 pairs taken at a distance of 1 month each with 1 day difference within the pairing dates
threshold_dates = [[pd.to_datetime('2015-07-31 01:00:00'), pd.to_datetime('2015-08-01 01:00:00')],
                   [pd.to_datetime('2015-08-31 01:00:00'), pd.to_datetime('2015-09-01 01:00:00')],
                   [pd.to_datetime('2015-09-30 01:00:00'), pd.to_datetime('2015-10-01 01:00:00')]]

# Note the above threshold date will prevent records in the training set from sharing time windows with the records in the test set.

test_results = []
models = []
for last_train_date, first_test_date in threshold_dates:
    # split out training and test data
    train_y = labeled_features.loc[labeled_features['datetime'] < last_train_date, 'failure']
    train_X = pd.get_dummies(labeled_features.loc[labeled_features['datetime'] < last_train_date].drop(['datetime',
                                                                                                        'machineID',
                                                                                                        'failure'],axis = 1), columns = ['model'])

    test_X = pd.get_dummies(labeled_features.loc[labeled_features['datetime'] > first_test_date].drop(['datetime',
                                                                                                       'machineID',
                                                                                                       'failure'],axis = 1), columns = ['model'])
```

```
[23] import lightgbm as lgb

    # Initialize the LightGBM model
    lgb_model = lgb.LGBMClassifier(random_state=42, objective='multiclass', verbose=2, boosting_type='gbdt') #
    lgb_model.fit(train_X, train_y)

    # Prepare test results DataFrame
    test_result_lgb = labeled_features.loc[labeled_features['datetime'] > first_test_date].copy()
    test_result_lgb['predicted_failure'] = lgb_model.predict(test_X)

    # Append results and model to lists
    test_results.append(test_result_lgb)
    models.append(lgb_model) # Runtime 17 seconds with T4 GPU
```

Feature Importance

```
plt.figure(figsize=(12, 4))
labels, importances = zip(*sorted(zip(test_X.columns, models[0].feature_importances_), reverse=True, key=lambda x: x[1]))
plt.xticks(range(len(labels)), labels)
_, labels = plt.xticks()
plt.setp(labels, rotation=90)
plt.bar(range(len(importances)), importances)
plt.ylabel('Importance')
plt.show()
```



Evaluation Metrics

```
from sklearn.metrics import confusion_matrix, recall_score, accuracy_score, precision_score, ConfusionMatrixDisplay

def Evaluate(predicted, actual, labels):
    output_labels = []
    output = []

    # Calculate and display confusion matrix
    cm = confusion_matrix(actual, predicted, labels=labels)
    print('Confusion matrix\n- x-axis is true labels (none, comp1, etc.)\n- y-axis is predicted labels')
    print(cm)

    # Calculate precision, recall, and F1 score
    accuracy = np.array([float(np.trace(cm)) / np.sum(cm)] * len(labels))
    precision = precision_score(actual, predicted, average=None, labels=labels)
    recall = recall_score(actual, predicted, average=None, labels=labels)
    f1 = 2 * precision * recall / (precision + recall)
    output.extend([accuracy.tolist(), precision.tolist(), recall.tolist(), f1.tolist()])
    output_labels.extend(['accuracy', 'precision', 'recall', 'F1'])

    # Calculate the macro versions of these metrics
    output.extend([[np.mean(precision)] * len(labels),
                   [np.mean(recall)] * len(labels),
                   [np.mean(f1)] * len(labels)])
    output_labels.extend(['macro precision', 'macro recall', 'macro F1'])

    # Find the one-vs.-all confusion matrix
    cm_row_sums = cm.sum(axis = 1)
    cm_col_sums = cm.sum(axis = 0)
    s = np.zeros((2, 2))
    for i in range(len(labels)):
        v = np.array([[cm[i, i],
                       cm_row_sums[i] - cm[i, i]],
                      [cm_col_sums[i] - cm[i, i],
                       np.sum(cm) + cm[i, i] - (cm_row_sums[i] + cm_col_sums[i])]])
        s += v
    s_row_sums = s.sum(axis = 1)

    # Add average accuracy and micro-averaged  precision/recall/F1
    avg_accuracy = [np.trace(s) / np.sum(s)] * len(labels)
    micro_prf = [float(s[0,0]) / s_row_sums[0]] * len(labels)
    output.extend([avg_accuracy, micro_prf])
    output_labels.extend(['average accuracy',
                          'micro-averaged precision/recall/F1'])

    # Compute metrics for the majority classifier
```

```
        # Compute metrics for the majority classifier
        mc_index = np.where(cm_row_sums == np.max(cm_row_sums))[0][0]
        cm_row_dist = cm_row_sums / float(np.sum(cm))
        mc_accuracy = 0 * cm_row_dist; mc_accuracy[mc_index] = cm_row_dist[mc_index]
        mc_recall = 0 * cm_row_dist; mc_recall[mc_index] = 1
        mc_precision = 0 * cm_row_dist
        mc_precision[mc_index] = cm_row_dist[mc_index]
        mc_F1 = 0 * cm_row_dist;
        mc_F1[mc_index] = 2 * mc_precision[mc_index] / (mc_precision[mc_index] + 1)
        output.extend([mc_accuracy.tolist(), mc_recall.tolist(),
                       mc_precision.tolist(), mc_F1.tolist()])
        output_labels.extend(['majority class accuracy', 'majority class recall',
                              'majority class precision', 'majority class F1'])

        # Random accuracy and kappa
        cm_col_dist = cm_col_sums / float(np.sum(cm))
        exp_accuracy = np.array([np.sum(cm_row_dist * cm_col_dist)] * len(labels))
        kappa = (accuracy - exp_accuracy) / (1 - exp_accuracy)
        output.extend([exp_accuracy.tolist(), kappa.tolist()])
        output_labels.extend(['expected accuracy', 'kappa'])

        output_df = pd.DataFrame(output, columns=labels)
        output_df.index = output_labels

        return output_df
```

```
[25] evaluation_results = []
     for i, test_result in enumerate(test_results):
         print('\nSplit %d:' % (i+1))
         evaluation_result = Evaluate(actual = test_result['failure'],
                                      predicted = test_result['predicted_failure'],
                                      labels = ['none', 'comp1', 'comp2', 'comp3', 'comp4'])
         evaluation_results.append(evaluation_result)
     evaluation_results[0]  # show full results for first split only
```

```
Split 1:
Confusion matrix
- x-axis is true labels (none, comp1, etc.)
- y-axis is predicted labels
[[71228    51    33     5    14]
 [   47   508     1     0     2]
 [   58     9   971     0     1]
 [   24     1     0   388     1]
 [   21     1     2     0   543]]
```