Springboard Capstone 3 Project Report

By

Krishnan G. Raghavan

Springboard Data Science

2024

Table of Contents

Abbreviations, Notations and Variables

This section lists some of the commonly used abbreviations, notations and variables in this report.

Abbreviations:

EDA. – Exploratory

Data Analysis

df. - Dataframe

Abstract

The use of machinery vibration and the technological advances that have been developed over the years, that make it possible to not only detect when a machine is developing a problem but to identify the specific nature of the problem for scheduled connection.

Fault detection and rotating machinery with the help of vibration sensors offers the possibility to detect damage to machines at an early stage and to prevent production down-times by taking appropriate measures.

In this study, vibration data from a shaft attached to a motor is obtained for different levels of unbalance load (mass and offset). The goal is to determine if such data can be used to predict worsening or damaged motor arrangement using machine learning.

Chapter 1. Introduction

With the growing demands of modern industries and increasing emphasis on efficiency, predictive maintenance is becoming a vital solution for machinery management, especially in sectors dependent on rotating machinery like manufacturing, energy, and transportation. According to a study by Deloitte, predictive maintenance can reduce breakdowns by 70%, lower maintenance costs by 25%, and reduce downtime by 35%.

Rotating machinery such as pumps, turbines, and compressors are critical components in various industrial sectors. When these machines fail, they can lead to costly operational disruptions and unplanned downtime. Traditionally, preventive maintenance has been employed, scheduling maintenance tasks at regular intervals based on time or usage. However, this often leads to unnecessary maintenance or missing unexpected failures.

Machine learning algorithms offer a more advanced solution by predicting machinery failure based on real-time data, such as vibration patterns, temperature, and operational speed. These algorithms can detect anomalies, such as bearing wear or shaft misalignment, that might go unnoticed with manual inspection, allowing for timely interventions that prevent costly failures. By integrating predictive algorithms, companies can optimize their maintenance strategies and avoid the expenses associated with sensor-heavy real-time monitoring systems.

1.1 Business Case

Problem: How can vibration data be used to predict possible damage to rotating machinery?

Customer's needs: Customer is an OEM who sells rotating machinery and wants to help their clients prevent unnecessary downtime and repair costs, hence increasing confidence in their product.

1.2 Such a Project is useful for...

- An OEM manufacturer of rotating machinery concerned about costs of downtime and repair.
- An operator of rotating machinery who wants to prevent unnecessary downtime and repair

1.3 Project Overview

In this project, we explore various ML modeling options to determine the feasibility of predicting rotor unbalance. We develop a custom metric. The salient features of this project are:

- Process for data extraction, exploration and processing with the goal of making predictive models for rotor unbalance
- Training and performance of the models
- Comparison of the models – development time and evaluation on test data

1.4 Overview of this Report

The following is an overview of the contents of this report:

- Chapter 2: Data Extraction, Exploration and Pre-Processing
- Chapter 3: Modeling Data Preparation, Model Creation, Model validation
- Chapter 4: Results – Model performance comparison
- Chapter 5: Discussion
- Chapter 6: Conclusions

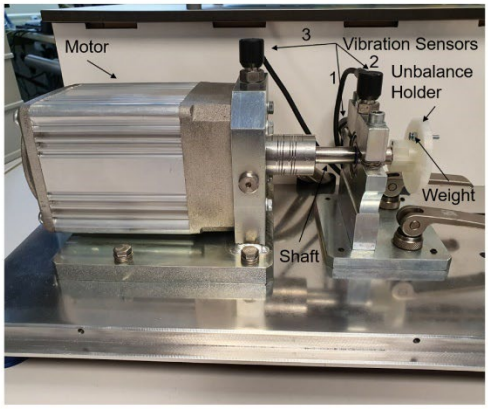
Chapter 2. Data Extraction, Exploration and Pre-Processing

In this chapter, we talk about what kind of data was used in the project, where it was obtained from and an overview of how it was used in this project to solve the problem.

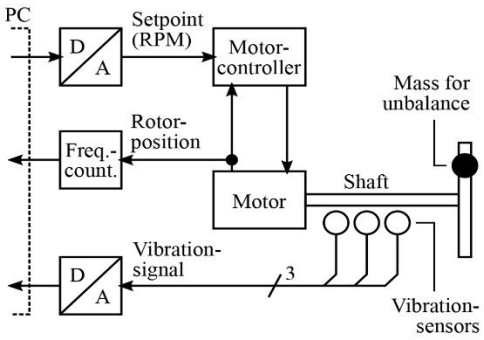
2.1 Data Acquisition

Data was acquired from the Kaggle Website [Vibration Analysis on Rotating Shaft], which contains data from vibration sensors attached to a shaft bearing. The following steps were involved in data extraction:

1. **Dataset Overview**: The dataset contains Motor voltage, RPM, unbalance level values and data from vibration sensors. The setup is shown in the figure below:



Measurement setup



Block diagram of the measurement setup

The setup for the simulation of defined unbalances and the measurement of the resulting vibrations is powered by an electronically commutated DC motor (WEG GmbH, type UE 511 T), which is controlled by a motor controller (WEG GmbH, type W2300) and is fixed to the aluminum base plate by means of a galvanized steel bracket. **Vibration sensors** (PCB Synotech GmbH, type PCB-M607A11 / M001AC) are attached to both the bearing block and the motor mounting and are read out using a 4-channel data acquisition system (PCB Synotech GmbH, type FRE-DT9837).

Using the setup described in above section, vibration data for unbalances of different sizes was recorded. The vibration data was recorded at a sampling rate of 4096 values per second. By varying the level of unbalance, different levels of difficulty can be achieved, since smaller unbalances obviously

influence the signals at the vibration sensors to a lesser extent.

In total, datasets for 4 different unbalance strengths were recorded as well as one dataset with the unbalance holder without additional weight (i.e. without unbalance). The rotation speed was varied between approx. 630 and 2330 RPM in the development datasets and between approx. 1060 and 1900 RPM in the evaluation datasets. Each dataset is provided as a csv-file with five columns:

1. V_in : The input voltage to the motor controller V_in (in V)

2. Measured_RPM : The rotation speed of the motor (in RPM; computed from speed measurements using the DT9837)

3. Vibration_1 : The signal from the first vibration sensor

4. Vibration_2 : The signal from the second vibration sensor

5. Vibration_3 : The signal from the third vibration sensor

Overview of the dataset components:

ID	Radius [mm]	Mass [g]
0D/ 0E	-	-
1D/ 1E	14 ± 0.1	3.281 ± 0.003
2D/ 2E	18.5 ± 0.1	3.281 ± 0.003
3D/ 3E	23 ± 0.1	3.281 ± 0.003
4D/ 4E	23 ± 0.1	6.614 ± 0.007

- To enable a comparable division into a **development dataset** and an **evaluation dataset**, separate measurements were taken for each unbalance strength, respectively.
- This separation can be recognized in the names of the csv-files, which are of the form "**1D.csv**": The **digit** describes the unbalance strength ("**0**" = **no unbalance**, "**4**" = **strong unbalance**), and the **letter** describes the **intended use** of the dataset ("**D**" = **development or training**, "**E**" = **evaluation**)

An important assumption is that the data is already time ordered.

2. **Data Loading**: The dataset was quite large (~3 GB). To work around this, two approaches were used:

    a. A custom code to down sample the data to every 10^{th} row, and save it into files marked as 'trunc'

    b. PySpark to read and analyze the data till a viable strategy for detailed processing could be developed.

2.2 Exploration

1. Read the development files using a function ‘basic_eda_spark’ that reads a csv or list of csvs into a PySpark dataframe. A ‘load’ column was added to indicate the corresponding level of unbalance derived from the filename

2. Print the column list, first and last 3, info, description and unique counts for each column and df using a function ‘basic_eda_spark’

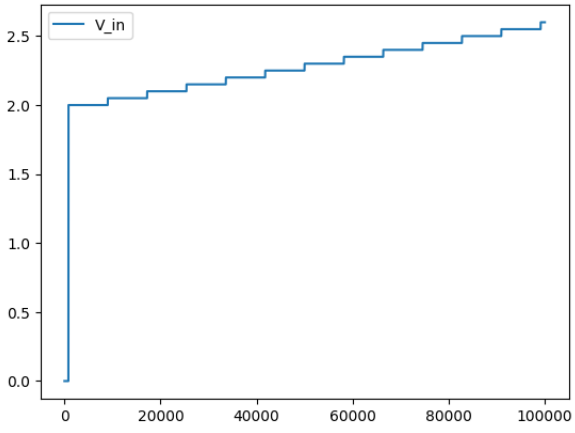
```
DESCRIPTION:
+-------+-----------------+------------------+-------------------+-------------------+-------------------+-----------------+
|summary|             V_in|       Measured_RPM|         Vibration_1|         Vibration_2|         Vibration_3|             load|
+-------+-----------------+------------------+-------------------+-------------------+-------------------+-----------------+
|  count|         13201553|          13201553|           13201553|           13201553|           13201553|         13201553|
|   mean|5.995143904660508|-35698.335014651355|0.001651138868151...|0.002609488510923...|0.004045518033946954|1.9993595450474653|
| stddev|2.327930769161795|  2986857.101551288| 0.05552156238148596|  0.0845743674615158|0.057237432687889736|1.4142643274997209|
|    min|              0.0|              -2.4E8|         -0.12001276|         -0.21576047|        -0.040333271|                0|
|    max|             10.0|          4091.723|          7.8491378|          8.7981558|          7.8299785|                4|
+-------+-----------------+------------------+-------------------+-------------------+-------------------+-----------------+
```

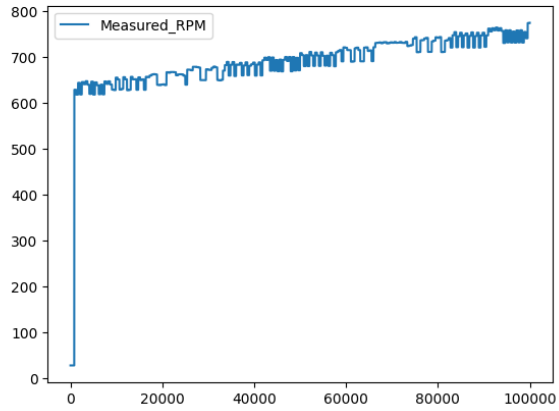
There are about 13 million total rows in the data

3. The summary and subsequent exploration showed RPM values of -2.4E8. Data was filtered for RPM values > 0.

4. There are many negative values of vibration. Since we are interested in the magnitude, we can get the absolute vibration values.

5. The following line plot were generated for the first 100,000 rows:

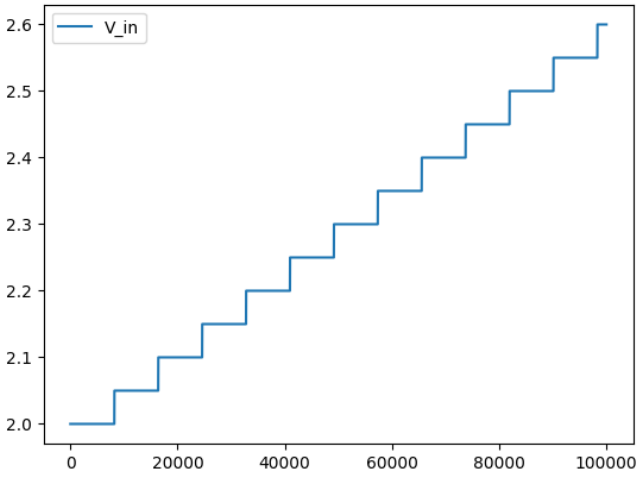


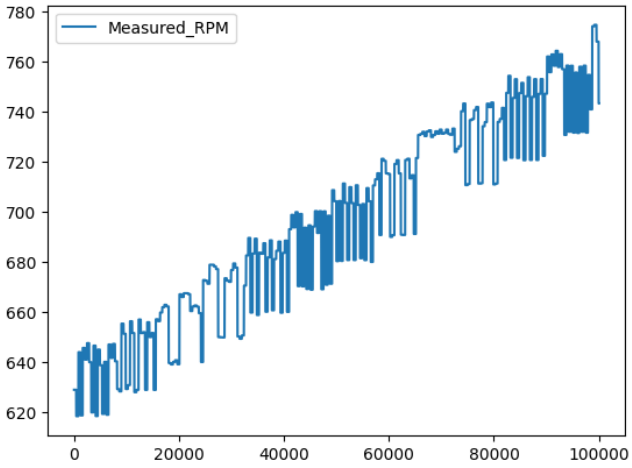


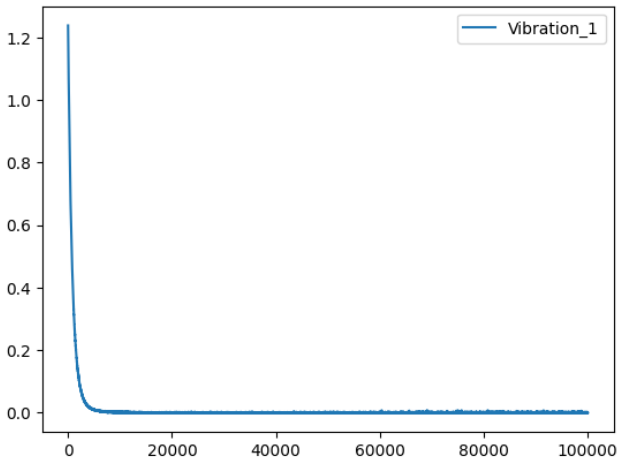
- The experiment seems to be designed around setting the motor voltage to specific levels and measure corresponding vibrations for different RPM and unbalance loadings. Therefore, we can possibly group by voltage (mean). However, doing so would also calculate the mean of load values across the voltages, which is not intended. Hence, we can include load in the grouping along with voltage.

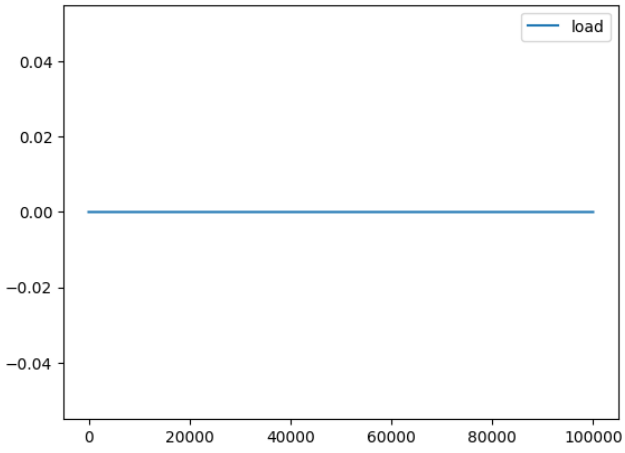
- Even for RPM>0, the expected operating range of the equipment seems to be > 600 rpm. Hence, we can filter for data with RPM > 600.

The following plots were generated:









There seems to be a strong relationship between voltage and RPM, and this is expected.

6. Grouping by Voltage (V_in) and load give us this summary:

summary	V_in	load	avg(V_in)	avg(Measured_RPM)	avg(Vibration_1)	avg(Vibration_2)	avg(Vibration_3)	avg(load)
count	805	805	805	805	805	805	805	805
mean	5.999999999999997	2.0	5.9999999999999992	1480.2197952421488	0.0063433936329156 32	0.0080400791577546 07	0.0043848541922359 18	2.0
stddev	2.325234701682919	1.4150927751172553	2.3252347016829047	492.42333510183096	0.0056657494151152 98	0.0094473253611200 8	0.0046804293726541 14	1.4150927751172553
min	2.0	0	2.0	636.9171586791892	5.1017552321461 09E-4	5.7834928147528E-4	0.0024570406092663 61	0.0
max	10.0	4	10.0	2332.1845971923567	0.0642699665010 2809	0.1108061556633 0182	0.0797962065775 0469	4.0

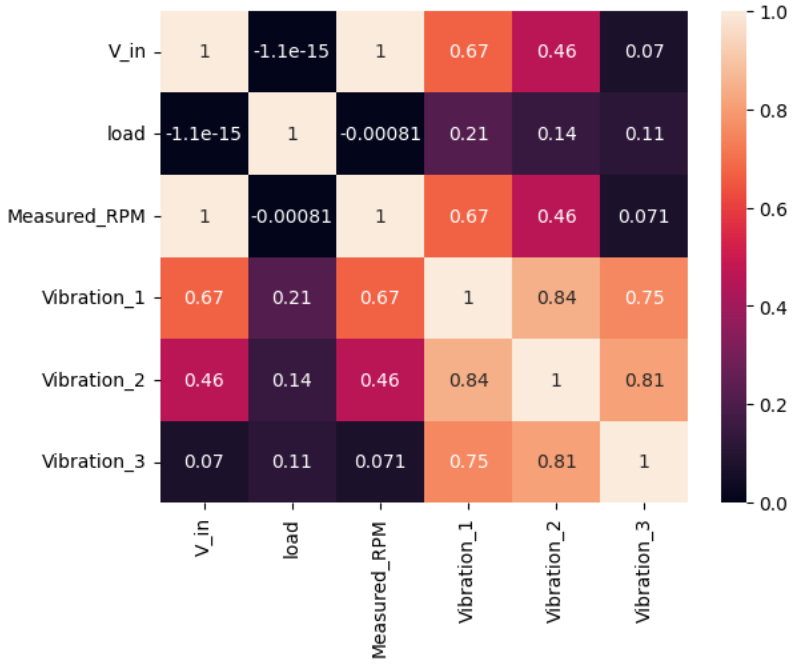
The V_in and load governments are repeated in the df. Also we don't need the 'avg' prefixes for the parameters. We also see that the df is now a lot more compact and easier to work with, with just 805 rows compared to the roughly 13 million earlier.

7. We can first call the groupBy action on the read transformation in PySpark, and convert the resultant to a pandas dataframe. Doing so and printing out the info:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 805 entries, 0 to 804
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   V_in          805 non-null    float64
 1   load          805 non-null    int32
 2   Measured_RPM  805 non-null    float64
 3   Vibration_1   805 non-null    float64
 4   Vibration_2   805 non-null    float64
 5   Vibration_3   805 non-null    float64
dtypes: float64(5), int32(1)
memory usage: 34.7 KB
```

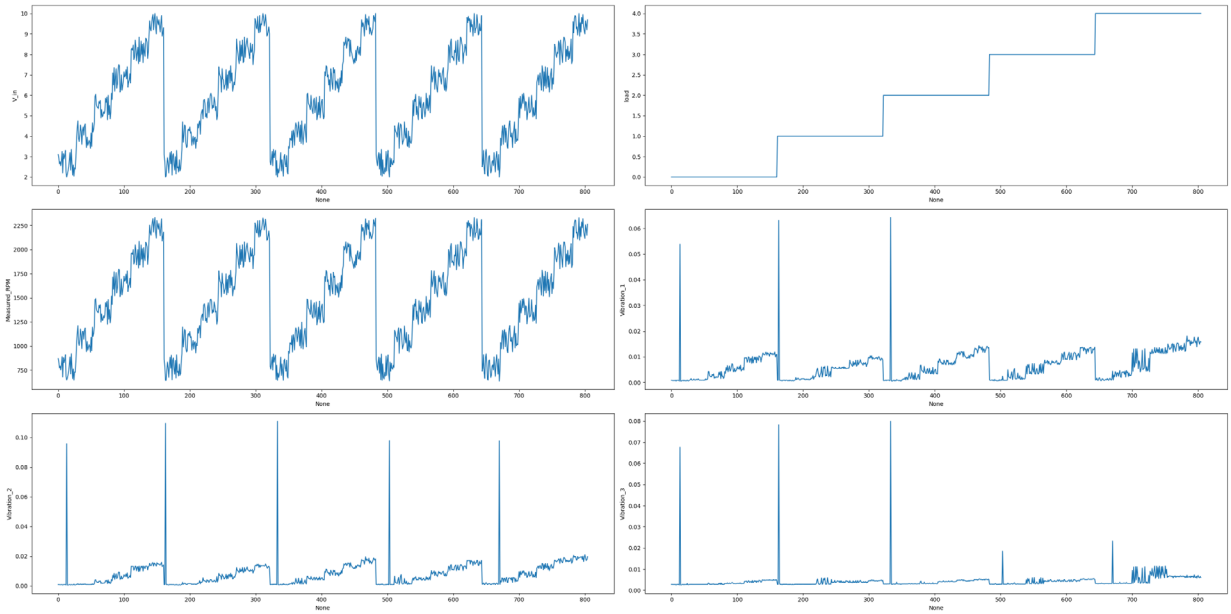
We have all non-null values of the required datatypes (notably load is int).

8. A heatmap of the variables is shown below:



We see that the vibration components are weakly related to load. The voltage and RPM components are very strongly related which we expect.

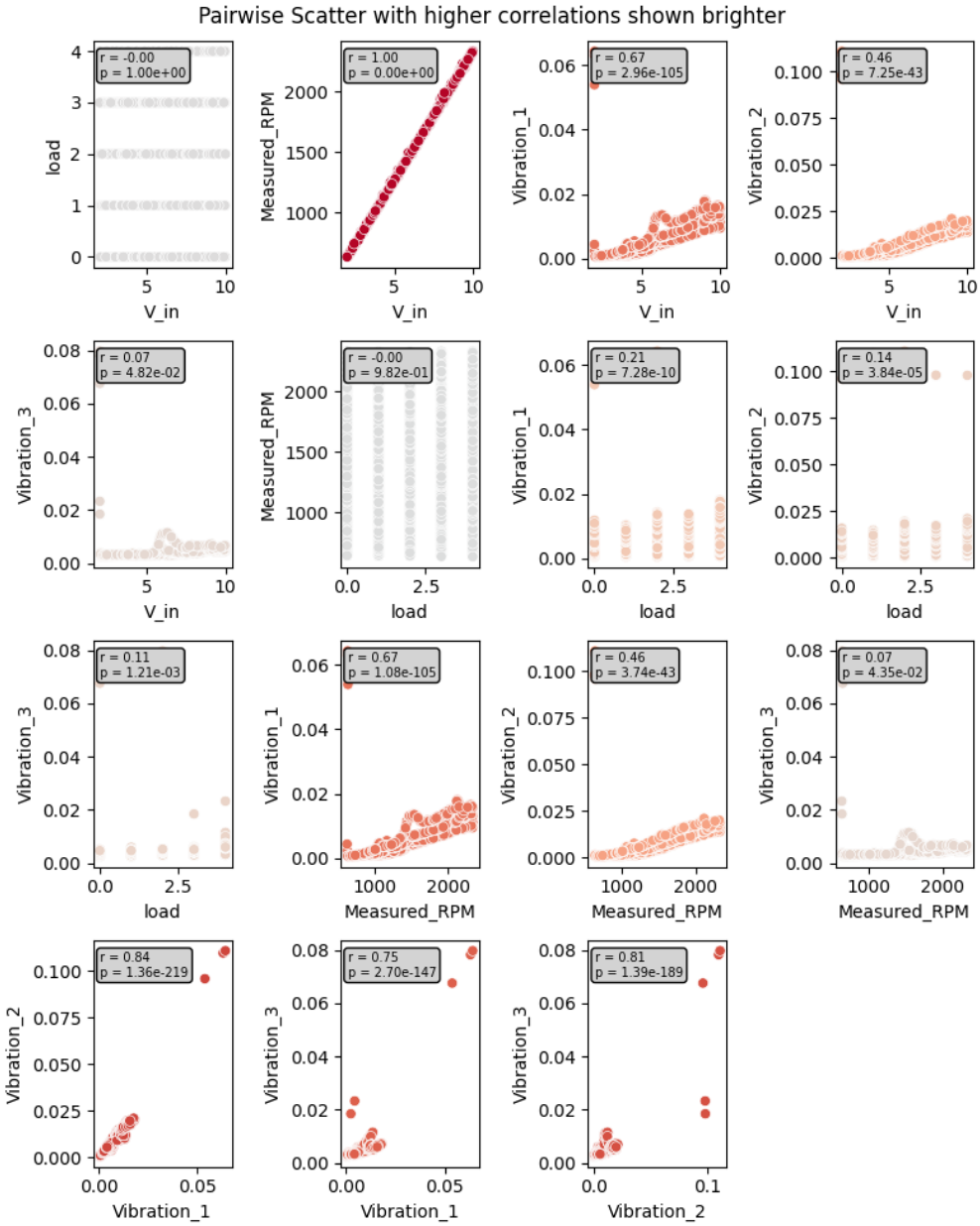
9. Plotting the line plots:



The data is consistent with the description of the experiment of varying RPM to observe

vibrations for different levels of loading.

10. Plotting scatter plots:

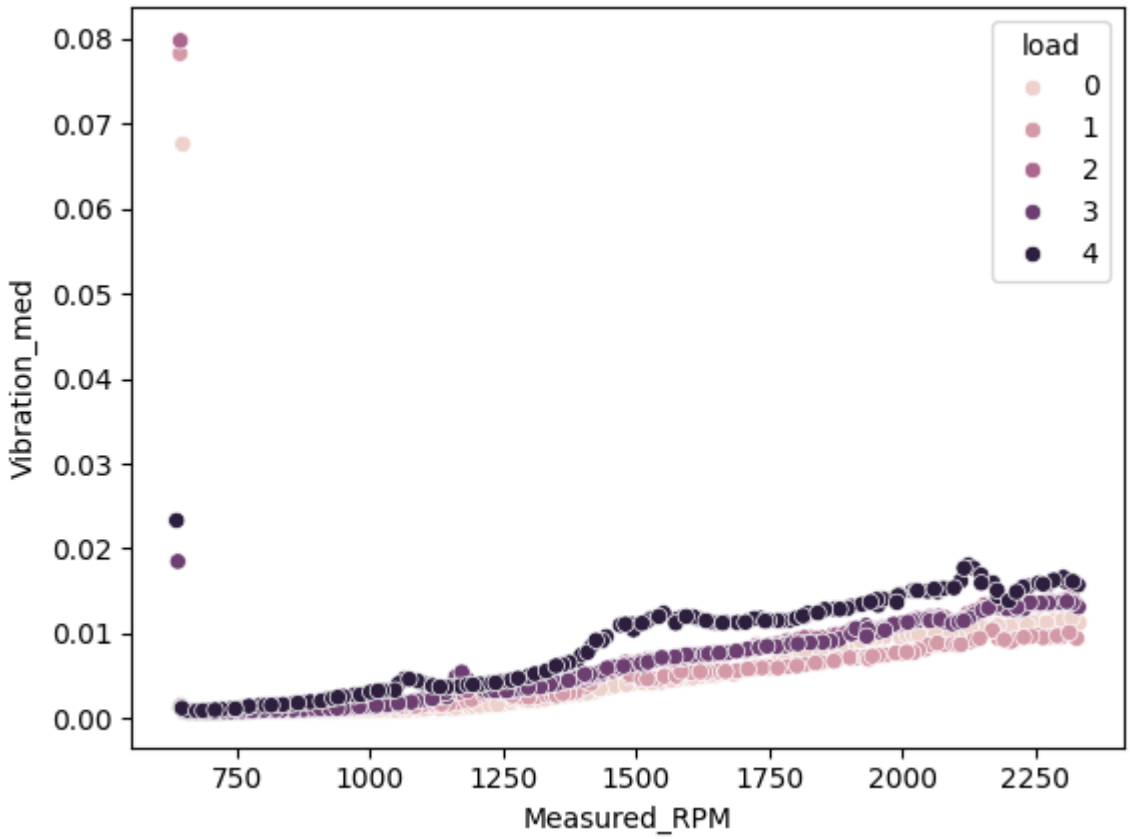


The vibration components are strongly correlated to each other. In order to reduce multicolliniearity, we can use a single parameter which is the median of the three vibration values.

The important parameters then are RPM (V_in is correlated), median vibration and loading

(target).

11. Plot of Vibration vs. RPM grouped by load:

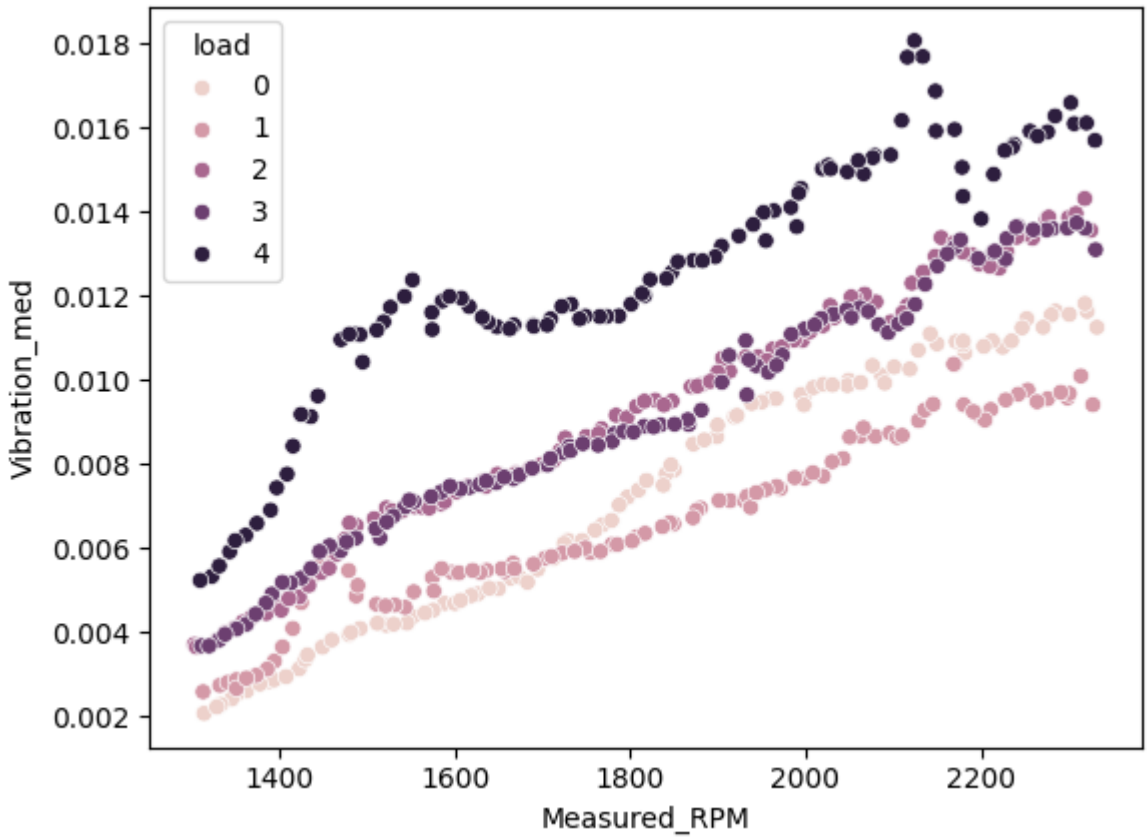


The vibrations do indeed rise as load increases, for a given RPM.

The effect is more pronounced between Loads 1, 2, and Loads 3-4.

Finally, we can infer that at RPMs below 1300 the trend is much stronger above this value. We also get rid of the cutoff vibration points at our RPM, which electrical issues may indicate

12. **This suggests that we can create a classification model to predict loading level using median vibration and RPM.**



There is barely any resolution between load levels 2 and 3. One would expect the number of true positions here to be fewer. Load level 0 shows more vibration than load level 1 at higher speeds—this is interesting, as one might expect the opposite. Increasing the weight of the load (load 4) has a drastic effect on the vibration characteristics.

2.3 Variable Selection

What variable to focus on? In this case, the answer is simply '**load**' since the load unbalance level is what we want to predict.

Chapter 3. Modeling Data Preparation, Model Creation, Model validation

This chapter discusses the process utilized to go from ‘Question to Answer’. It elaborates on how the data is transformed to analyze the target variable and the various analysis steps.

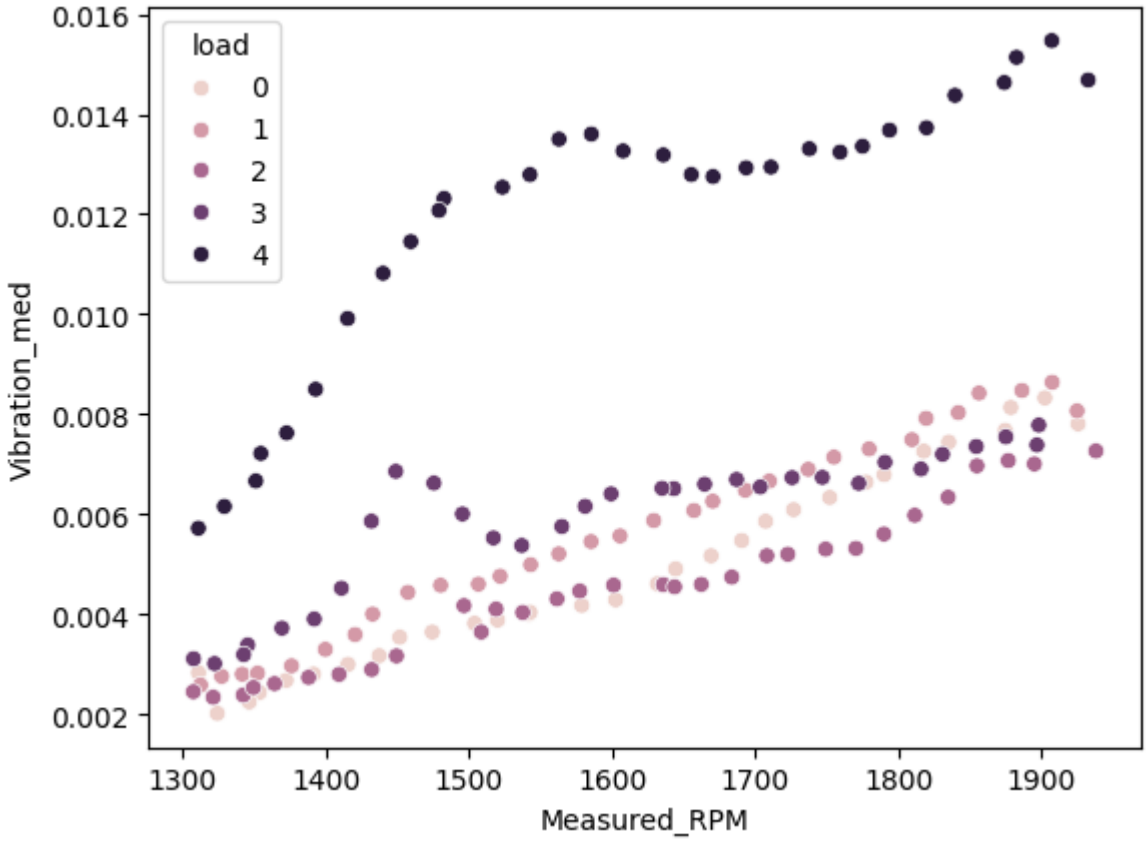
3.1 Data Preparation

Summarizing the data reading and transformation process:

1. Add a 'load' column corresponding to level of unbalance.
2. Filter for RPM > 1300.
3. Take absolute values of vibrations.
4. Group by 'V_in' and 'load' and remove the additional columns.
5. Calculate median of vibrations 1, 2, 3.
6. Select 'Measured_RPM', 'Vibration_med', and 'load' columns.

We perform these steps on the test data as well (dataset ‘E’).

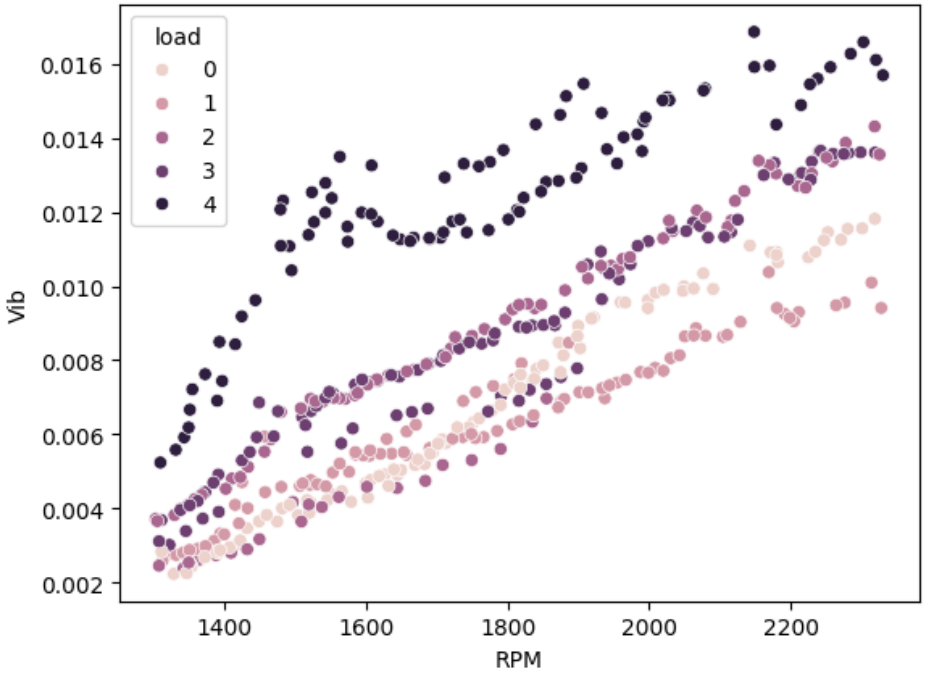
Making similar characteristic plots for the evaluation data:



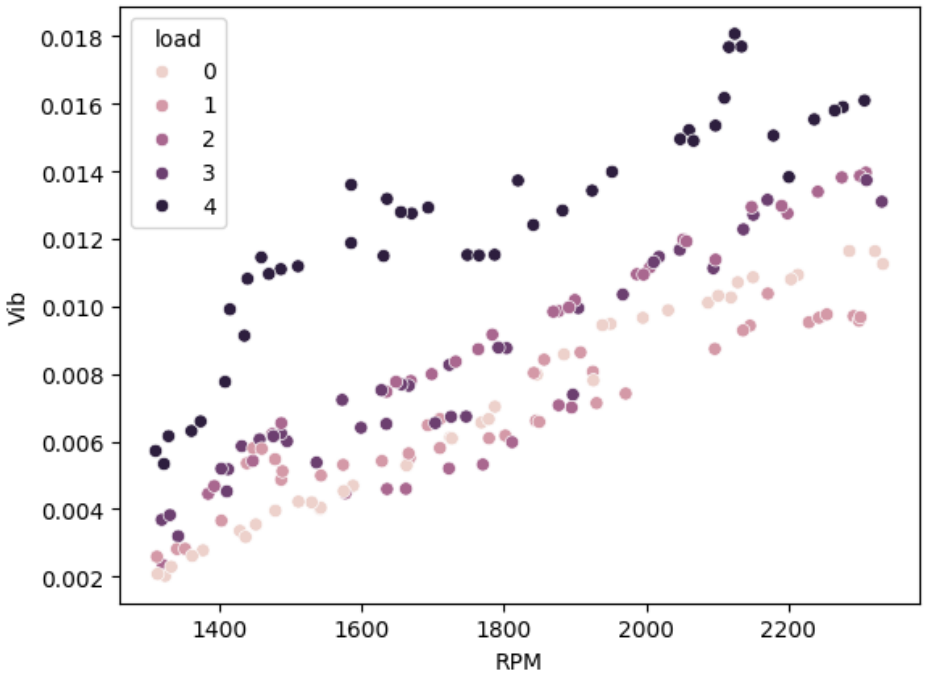
The resolution between lines for the training and test sets is very different. The lower levels of loading in the test set are barely distinguishable. This suggests that the train set may not be a good representation of the population, or that the test set is not representative of the population. To obtain a better representation, we could try to split the entire data (train+test) and obtain new train and test sets by random shuffling.

Doing so using the train_test_split with a train_size of 0.7, and plotting the training and evaluation sets, we get:

Training characteristics:



Evaluation characteristics:



- The train and evaluation data now have more similar characteristics

- This suggests that there could be some unnoticed sampling bias in the original distribution offered by the authors (as 'D' and 'E' sets). This seems to be addressed to some extent by recombining and shuffling.

3.2 Model Creation and Performance

Favorable: The higher load levels are accurately predicted to prevent imminent damage (TP) - Sum of [(3,3), (4,4)].
Unfavorable 1: The higher load levels are predicted as lower loads (FN) - Sum of [(0,3), (1,3), (0,4), (1,4)]
Unfavorable 2: The lower load levels are mistaken as higher loads leading to false flags - Sum of [(3,0), (3,1), (4,1), (4,2)]

Where (a,b):(Predicted,True)
.

Based on the above, the following metric was used:

Model_score = favorable / (favorable + unfavorable_1 + unfavorable_2)

Model Score > .7

Model Pipeline:

The model pipeline consisted of:

1. 'Scaler': MinMaxScaler to bring the different orders of magnitude of RPM and vibrations to a similar range (0-1). This was preferred over standard scaling since the data is not normally distributed (or expected to be)

2. 'Model': A selection of different types of models (RandomForestClassifier, K-Neighbors, XGB Classifier and Logistic Regressor) with their hyperparameters.

Grid Search:

The models were trained on a grid search by dividing the train data into 10 cross-validation folds. Model with different hyperparameter settings were trained across 9 of these folds and evaluated on the 10^{th}, and the model with the best score was returned for evaluation of the test set.

Confusion Matrix:

A confusion matrix with the percentage of the prediction of each class is created

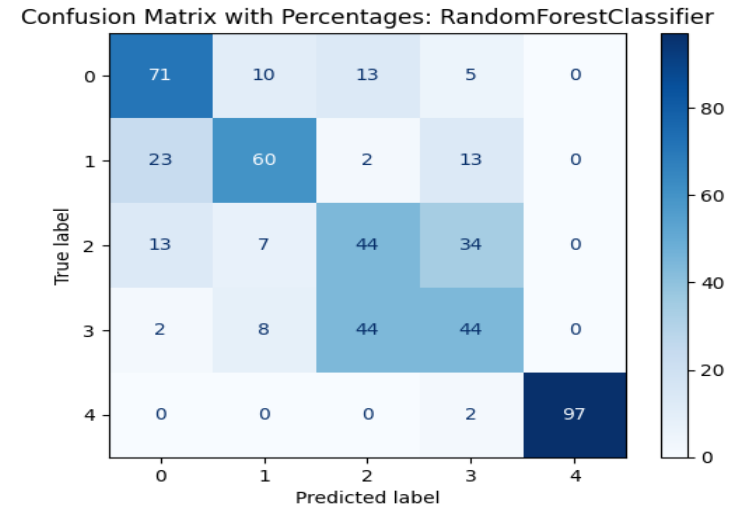
Classification Report:

A report of different classifier properties is printed. Including precision, recall, f1 score for each of the classes of unbalance (0, 1, 2, 3 and 4).

Random Forest Classifier Model:

Random Forest Classifier is a supervised machine learning algorithm that operates by creating an ensemble of decision trees. Each tree is trained on a random subset of the data (with replacement) and a random subset of the features. During prediction, the algorithm aggregates the results from all the trees (typically through majority voting for classification tasks) to make the final prediction

```
RandomForestClassifier
Best Score, Best Params:  0.6251515151515152 {'Model__max_depth': 15, 'Model__n_estimators': 50}
```



	precision	recall	f1-score	support
0	0.642857	0.710526	0.675000	38.000000
1	0.696970	0.605263	0.647887	38.000000
2	0.435897	0.447368	0.441558	38.000000
3	0.432432	0.444444	0.438356	36.000000
4	1.000000	0.976190	0.987952	42.000000
accuracy	0.645833	0.645833	0.645833	0.645833
macro avg	0.641631	0.636759	0.638151	192.000000
weighted avg	0.651277	0.645833	0.647519	192.000000

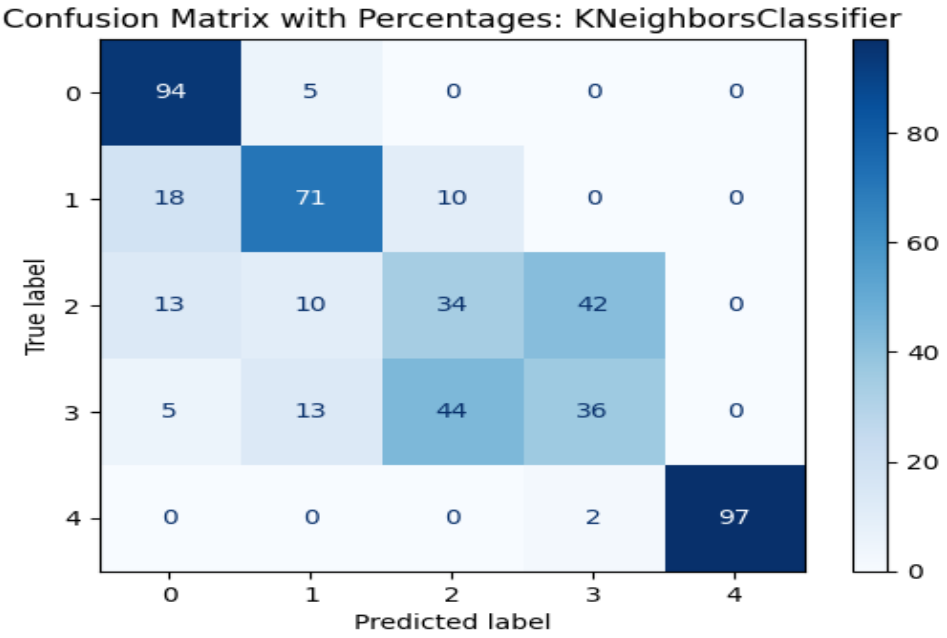
```
Model Score (Higher is better): 0.8382
```

K-Neighbors Classifier:

The KNeighbors Classifier is a type of supervised machine learning algorithm used for both classification and regression tasks, but it's most applied in classification problems. It is part of the k-nearest neighbors (k-NN) algorithm family, which works by classifying data points based on their proximity to other data points in the training set.

How It Works:

- Data Point Classification: When a new data point is introduced, the classifier looks for the k closest points (neighbors) in the training dataset, based on a distance metric (e.g., Euclidean distance).
- Majority Voting: The algorithm assigns the new point to the class that is most common among its k nearest neighbors.
- No Training Process: k-NN does not have a traditional training phase, as it stores all the training data and performs classification at the prediction stage by checking the nearest neighbors.



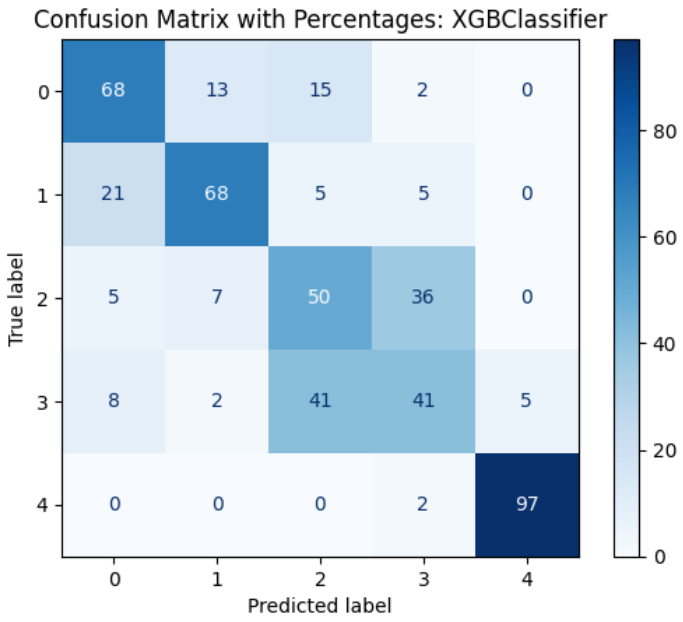
	precision	recall	f1-score	support
0	0.720000	0.947368	0.818182	38.000000
1	0.710526	0.710526	0.710526	38.000000
2	0.393939	0.342105	0.366197	38.000000
3	0.433333	0.361111	0.393939	36.000000
4	1.000000	0.976190	0.987952	42.000000
accuracy	0.677083	0.677083	0.677083	0.677083
macro avg	0.651560	0.667460	0.655359	192.000000
weighted avg	0.661092	0.677083	0.665011	192.000000

Model Score (Higher is better): 0.8852

XGB Classifier:

XGBClassifier is an implementation of the eXtreme Gradient Boosting (XGBoost) algorithm for classification tasks, which is part of the XGBoost library. XGBoost is an optimized, scalable version of gradient boosting that has been widely adopted in machine learning for its speed, performance, and flexibility. It works by combining an ensemble of decision trees to produce a strong predictive model.

```
XGBClassifier
Best Score, Best Params:  0.5961111111111111 {'Model__booster': 'gbtree', 'Model__n_estimators': 100}
```



	precision	recall	f1-score	support
0	0.666667	0.684211	0.675325	38.000000
1	0.742857	0.684211	0.712329	38.000000
2	0.452381	0.500000	0.475000	38.000000
3	0.454545	0.416667	0.434783	36.000000
4	0.953488	0.976190	0.964706	42.000000
accuracy	0.661458	0.661458	0.661458	0.661458
macro avg	0.653988	0.652256	0.652428	192.000000
weighted avg	0.662305	0.661458	0.661201	192.000000

```
Model Score (Higher is better): 0.8889
```

Logistic Regression:

Logistic Regression is a supervised machine learning algorithm. Despite its name, logistic regression is a classification algorithm, not a regression algorithm. It is based on the concept of the logistic function (also called the sigmoid function), which maps predicted values to probabilities.

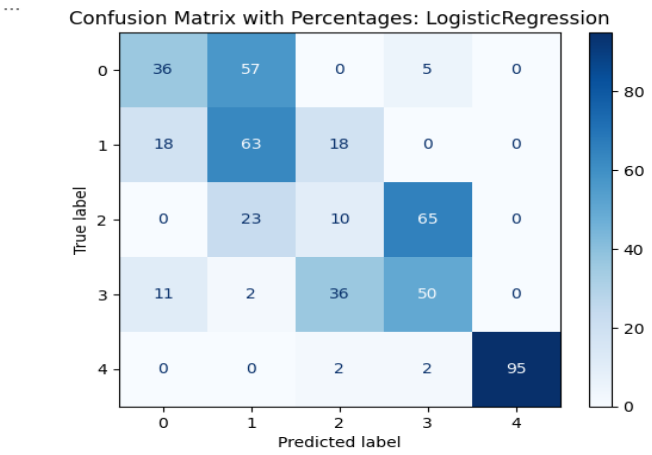
How Logistic Regression Works:

1. **Linear Combination**: Logistic regression calculates a linear combination of the input features, similar to linear regression:

z=β0+β1x1+β2x2+⋯+βnxnz = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_nz=β0+β1x1 +β2x2+⋯+βnxn

Where x_1, x_2, ..., x_n are the input features, and β_0, β_1, ..., β_n are the learned weights.

2. **Prediction**: The linear combination z is passed through the sigmoid function to predict the probability of the outcome.
3. **Cost Function**: Instead of using the mean squared error (as in linear regression), logistic regression uses the **log-loss (or cross-entropy loss)** as the cost function, which penalizes wrong predictions based on the probability output.
4. **Optimization**: The model is trained by minimizing the log-loss cost function using optimization techniques like **gradient descent**.



	precision	recall	f1-score	support
0	0.560000	0.368421	0.444444	38.000000
1	0.428571	0.631579	0.510638	38.000000
2	0.160000	0.105263	0.126984	38.000000
3	0.391304	0.500000	0.439024	36.000000
4	1.000000	0.952381	0.975610	42.000000
accuracy	0.520833	0.520833	0.520833	0.520833
macro avg	0.507975	0.511529	0.499340	192.000000
weighted avg	0.519441	0.520833	0.509891	192.000000

Model Score (Higher is better): 0.8788

3.3 Model Comparison

The below table shows the model training time and RMSE:

Model	Score
RF	.84
KNN Classifier	.89
XGB Classifier	.89
Logistic Regressor	.88

Although KNN, XGB and Logistic Regressor give very similar scores and better than RF, indicating all reasonable models, we can use a combination of model outputs for best results:

1. While KNN has significantly higher accuracy than others at level 0 (no unbalance), XGB outperforms KNN at almost all other levels, while managing to not falsely predict higher levels. Due to its higher accuracy at level 0, KNN can be used to overrule XGB if it predicts a zero.

2. XGB regressor is better than others in the critical region of classes 2 and 3, where the predictions are poor (due to low resolution as seen before). Hence XGB can be used for the other levels.

3. Logistic Regressor has poor TP at level 2

Hence, we can use the XGB regressor but consider KNN for the case of no unbalance prediction

Chapter 4. Conclusions

We've obtained and transformed the data and used different models to try and predict the unbalance levels for a rotating equipment.

Based on the model summary, we find the XGB Classifier to be the more reliable choice due to its score and its capacity to not overpredict level 2 unbalance. It also meets the benchmark of 70% accuracy we set in the proposal. However, the KNN model performs better at lower loads.

This makes the XGB Classifier model choice for the task. The KNN model can be used to complement for predicting a shaft with no unbalance.

Chapter 5. Discussion

While performing the data analysis and modeling, there were several challenges to overcome as well as identifying some areas of future work.

5.1 Challenges

1. Due to the very similar Vibration vs. RPM characteristics of levels 2 and 3, it was almost impossible to resolve them enough to classify them properly.

2. Vibration levels for load level 0 exceeds that for load level 1. This is not expected and should be explored further

5.2 Suggestions for Improvement

1. Explore reasons for why the level 0 vibrations exceed those of level 1 at higher RPMs. Probably a pre-existing unbalance? This might call for a repeat of the experiments depending on the findings

2. Vibration sensors can be placed at different radial orientations to see if it helps with getting more information to make better predictions especially at difficult to resolve levels such as 2 and 3.

3. Additional data techniques such as smoothing, in combination with more involved modeling such as neural networks might also help give better resolution ability.