

Identifying the Type of Finished Product Based on Product Elements Using Machine Learning

A Partnership Project

with

Ernst and Young (EY)

Sujatha Ramesh

Matric Number: 30006034

10th June 2022

Table of Contents

Executive Summary	3
1 Introduction.....	4
1.1 Motivation	5
1.2 Business Problem	5
2 Methodology	6
2.1 Understanding the Business Problem	6
2.2 Data Collection	6
2.3 Data Cleaning and Data Pre-Processing.....	7
2.4 Feature Engineering	9
2.5 Model Selection.....	12
3 Results.....	13
4 Conclusion and Future Work	14
5 References	15

Executive Summary

The goal of this project is to use machine learning approaches to assign the correct product to an incomplete Bill of Material using the product elements. The dataset used for the project consist of 341 samples of different Bill of Materials containing subcomponents and their respective finished products from a dairy processing industry (GEA). Several data pre-processing and feature engineering steps can be done on the dataset to ensure that the chosen algorithm performs to its optimum capability. An algorithm can be established to select all finished products that contained all product elements in their construct from a given list of subcomponents in a bill of material. This brings out all the likely products that the given Bill of Material component list could represent. Hence, the next step of this project is to optimize and determine which is the likely product? From understanding of the business problem, it is determined that one of the best models that gives an optimal solution would be the Naive Bayes Algorithm. After the Naive Bayes Algorithm is implemented, the most likely finished product from the given list of components can be decided and obtained.

1 Introduction

A Bill of Materials (BOM) is a list of all necessary materials and components required to make a product. The data structure of each product in the factory can be described by the BOM. It describes the parent-child relationship of the items required for a specific item. It includes not only the fundamental properties of materials, suppliers, processes, and so on, but also the number of components required for each part of the relationship. The BOM is a data structure that is shaped like a tree. The BOM in the factory connects the entire factory product's life cycle. It usually appears in a hierarchical format, with the highest level displaying the finished product and the bottom level showing individual components and materials (Zhou & Cao, 2019) .

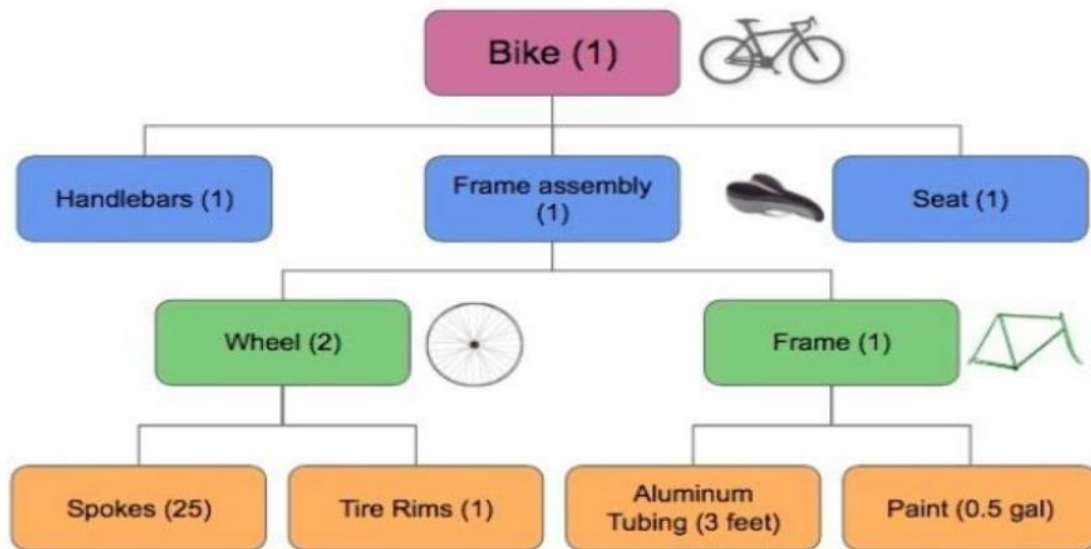


Figure 1: Sample of Bill of Material Structure (OpenBOM (openbom.com), 2018)

Manufacturing bills of materials and engineering bills of materials are the two main types of BOMs. A manufacturing bill of materials lists all the assemblies and parts that go into making a final product that can be shipped. It also includes the materials needed to transport the goods to the client in its packaging. It contains processes that must be implemented on the product before it can be finished, as well as all the information needed for manufacturing activities. The design of the finished product is defined by an engineering bill of materials. All alternative and substitute part numbers and parts in the drawing notes are included. The product code, part name, part number, part revision, description, quantity, unit of measure, size, length, weight, and specifications or characteristics of the product are all listed on each line of the bill of materials (Grant, 2022).

1.1 Motivation

Any company that wishes to take its manufactured items from concept to customer with ease, starts with a detailed manufacturing Bill of Materials (BoM). Assume a company enters production with a bill of materials that is incorrect or missing critical information or instructions. The production work could cause significant delays or perhaps halt the entire manufacturing process. A poorly prepared Bill of Materials can be extremely costly to a company, in terms of time delays, capital and human resources (Dear Systems, 2021) .

As a result, BOM Analytics is critical to the manufacturing industry for a variety of reasons:

- Proactively manage component obsolescence and compliance.
- Avoid essential component shortages to fulfill production and sales targets.
- Eliminate redundant components, suppliers, and surplus inventories.
- Avoid missed opportunities in areas where environmental compliance laws apply.
- Manage extended supply chains and eliminate counterfeit part concerns.
- Identify technology advances that can alter designs, sources, or margins across the product cycle.

1.2 Business Problem

How can we identify the type of the finished product based on the product elements, using Machine Learning?

2 Methodology

2.1 Understanding the Business Problem

As the risks of obsolescence, counterfeiting, and non-compliance continue to rise, managing components and suppliers is more important than ever. A successful BOM analysis system will enable businesses to accelerate the introduction of new products, minimize costly production disruptions or product redesigns, and improve product sustainability over longer service lives (IHS Markit, 2016).

In detail, many of the finished products in advance manufacturing sector have lot of sub-components that are commonly shared. So, in approaching to solve this problem, we defined a clear goal which is to correctly identify the finished product from bill of material containing different components and product elements.

2.2 Data Collection

The data used for this analysis is an extract collected in an excel format from a dairy processing industry (GEA), a client of Ernst and Young (EY). BoM data is classified into two types: Single level and Multi level. In this project, we had access to a single level BoM data with only two layers – End Product and subcomponents.

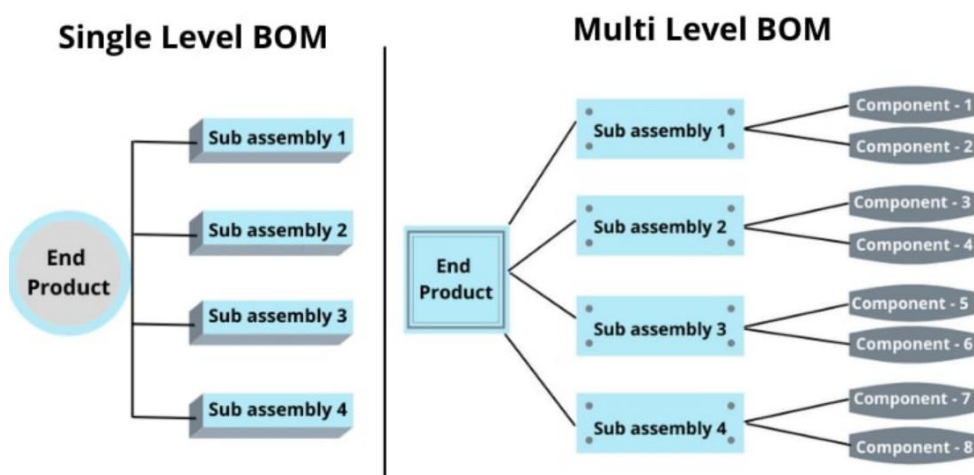


Figure 2: Single Level and Multi Level BOM

The dataset had a total of 341 data samples which contained the following variables:

- **STLNR**: refers to BoM unique ID
- **MATNR**: Header of the BoM (Parent article)
- **MAKTX**: Header description (Name of the article)

- **MTART**: Material Type (KMAT refers to configured material or produced finished good)
- **MATKL**: Material class (category)
- **GROES**: Material Size
- **PRDHA**: Product Structure Code (unique code, referring to the product structure classes)
- **MEINS**: Base unit of measure (ST refers to piece)
- **IDNRK**: BoM Item (Child article)
- **STPO_MTART**: Material type of BoM Item
- **STPO_MATKL**: Material class of BoM Item
- **STOP_PRDHA**: Product Structure Code of BoM Item
- **STOP_MAKTX**: BoM Item Description.

2.3 Data Cleaning and Data Pre-Processing

The transformation of the raw dataset into a comprehensible format is known as data preprocessing. Data preprocessing is a critical step in data mining that improves data efficiency. Data preprocessing procedures have a direct impact on the results of any analytic algorithm. All pre-processing steps were implemented in python with the following steps:

- Imported the dataset and libraries needed for the analysis

```
#Import all libraries used in Analysis
import numpy as np # linear algebra
import pandas as pd
import matplotlib.pyplot as plt #visualization
import seaborn as sns

import math
import random
import json

import warnings
warnings.simplefilter(action="ignore", category=FutureWarning)

from scipy.stats import norm
from sklearn.preprocessing import MinMaxScaler
from scipy import stats
from sklearn.naive_bayes import MultinomialNB
```

Figure 3: Import Libraries

```
df= pd.read_excel('/content/Bom Analytics.xlsx')
```

```
[3] # Check Data Structure
df
```

	STLNR	POSNR	MATNR	MAKTX	MTART	MATKL	GROES	PRDHA	MENGE	MEINS	IDNRK	STPO_MTART	STPO_MATKL	STPO_PRDHA	STPO_MAKTX
0	364014	110	79101000040	Side by Side Parlour	KMAT	Z	SbS	1001040399	1	ST	7.910100e+10	KMAT	Z	1.004040e+09	Milk Pipe
1	364014	240	79101000040	Side by Side Parlour	KMAT	Z	SbS	1001040399	1	ST	7.910100e+10	KMAT	Z	1.004040e+09	Milk Pipe
2	364014	185	79101000040	Side by Side Parlour	KMAT	Z	SbS	1001040399	1	ST	7.910101e+10	KMAT	Z	1.008070e+09	Milking Control Unit EasyDe
3	364014	400	79101000040	Side by Side Parlour	KMAT	Z	SbS	1001040399	1	ST	7.910101e+10	KMAT	Z	1.008070e+09	Milking Control Unit EasyDe
4	503691	20	79101000040	Side by Side Parlour	KMAT	Z	SbS	1001040399	1	ST	7.910100e+10	KMAT	Z	1.004040e+09	Milk Handling
...
336	406751	80	79101001056	Herd Management	KMAT	Z	MIone	1001050398	1	ST	7.160584e+10	SKU	JG	7.100000e+09	Slip-on Sleeve
337	406751	40	79101001056	Herd Management	KMAT	Z	MIone	1001050398	1	ST	7.160584e+10	SKU	JG	7.100000e+09	Slip-on Sleeve
338	406751	10	79101001056	Herd Management	KMAT	Z	MIone	1001050398	1	ST	NaN	NaN	NaN	NaN	NaN
339	406751	30	79101001056	Herd Management	KMAT	Z	MIone	1001050398	1	ST	NaN	NaN	NaN	NaN	NaN
340	406751	5	79101001056	Herd Management	KMAT	Z	MIone	1001050398	1	ST	NaN	NaN	NaN	NaN	NaN

341 rows x 15 columns

Figure 4: Import Dataset

- Dealing with Missing Values and Datatypes:** As a next step, this project evaluated the missing values within the data set to better understand how to deal with them. First, we search for patterns within the variables through visual inspection. There, we speculate how the distribution of the missing values would look like if they were available by discussing if the missing data is “missing completely at random”, “missing at random” or “not missing at random.” From understanding the business problem, we decided to drop the rows with NA values. Also, data types are checked, to ensure that accuracy in the analysis. The fields were changed to strings and unnecessary preceeding zeros were removed from the IDNRK, MATNR and STPO_PRDHA columns.

```
# Check for missing values
df.isna().sum()
```

```
STLNR      0
POSNR      0
MATNR      0
MAKTX      0
MTART      0
MATKL      0
GROES      0
PRDHA      0
MENGE      0
MEINS      0
IDNRK     14
STPO_MTART 14
STPO_MATKL 14
STPO_PRDHA 14
STPO_MAKTX 14
dtype: int64
```

```
[57] # Change datatypes to string
df['IDNRK'] = df['IDNRK'].astype("string")
df['STLNR'] = df['STLNR'].astype("string")
df['MATNR'] = df['MATNR'].astype("string")
```

```
# Extract Data
df_bom = df.loc[:, ['STLNR', 'MATNR', 'IDNRK']]

# Remove all rows with NA
df_bom = df_bom.dropna()
```

Figure 5: Data Cleaning Steps

2.4 Feature Engineering

Feature engineering is the process of transforming raw data into features that better represent the underlying problem to the predictive models, resulting in improved model accuracy on unseen data (Browniee, 2014).

In this project, we had to restructure our data to accurately address our business problem. Here, are the following feature engineering steps that were done in this project.

- **Feature Selection:** From the understanding of the problem and acquiring necessary domain knowledge, we selected only to relevant features to conduct our analysis - STNLR, MATNR & IDNRK.

	STNLR	MATNR	IDNRK
0	364014	79101000040	79101004040
1	364014	79101000040	79101004040
2	364014	79101000040	79101008050
3	364014	79101000040	79101008050
4	503691	79101000040	79101004100
...
333	406751	79101001056	71605837220
334	406751	79101001056	71605837230
335	406751	79101001056	71605837240
336	406751	79101001056	71605837250
337	406751	79101001056	71605837210

327 rows × 3 columns

Figure 6: Features Selected for Analysis

- **Created New Feature (IDNRK_Count):** Next, we derived a new feature from the IDNRK_Count which is the frequency of occurrence of the subcomponents (IDNRK) in a finished product for a specific BOM. This is important as it is needed to solve the business problem.

	STNLR	MATNR	IDNRK	IDNRK_count
0	364014	79101000040	40531001042	1
1	364014	79101000040	79101001040	1
2	364014	79101000040	79101001041	2
3	364014	79101000040	79101001043	1
4	364014	79101000040	79101001044	1
...
212	630348	79101000059	79101010124	2
213	630348	79101000059	79101010125	2
214	630348	79101000059	79102004060	2
215	630348	79101000059	79102004061	2
216	630348	79101000059	79102006080	2

217 rows × 4 columns

Figure 7: New Feature IDNRK_count added to the dataset

- **Logic to get all products from a given subcomponent list:** In identifying the type of the finished product, a logic was created which helps select all products that have all the given subcomponents in its product construct.

For example, here is a given list of subcomponents which is the input.

```
component_list = ["79101004040", "79101004040", "79101004040", "79101004040", "79101004042", "79101004042", "79101004042", "79101004042",
"79101003050", "79101003070", "79101004010", "79101004010", "79101004010", "79101004010", "79101004010",
"79101004020", "79101004030", "79101006010", "79101008010", "79101008030", "79101003080", "79101005010",
"79101005020", "79101007010", "79101009010", "79101008020", "79101999000",
"79101005060", "79101004012", "79101004031", "79101003030", "79101003020", "79101003010",
"79101003040", "79101003065", "79101004040", "79101004042", "79101003050", "79101003070",
"79101004010", "79101004020", "79101004030", "79101006010", "79101008010", "79101008030",
"79101003080", "79101005010", "79101005020", "79101007010", "79101009010", "79101008020",
"79101999000", "79101005060", "79101004012", "79101004031", "79101003030", "79101003020", "79101003010", "79101003040", "79101003065"]
```

Figure 8: Sample of a given component list

The logic then brings as an output all the products that have these subcomponents in their product construct.

```
{
  "364014": {
    "product": "79101000040",
    "components": {
      "40531001042": 1,
      "79101001040": 1,
      "79101001041": 2,
      "79101001043": 1,
      "79101001044": 1,
      "79101001045": 1,
      "79101001046": 1,
      "79101001047": 1,
      "79101003010": 2,
      "79101003020": 2,
      "79101003030": 2,
      "79101003040": 1,
      "79101003050": 2,
      "79101003065": 1,
      "79101003070": 2,
      "79101003080": 2,
      "79101004010": 2,
      "79101004012": 2,
      "79101004020": 2,
      "79101004030": 2,
      "79101004031": 1,
      "79101004040": 2,
      "79101004042": 2,
      "79101005010": 2,
      "79101005020": 2,
      "79101005060": 2,
      "79101006010": 2,
      "79101007010": 2,
      "79101008010": 2,
      "79101008020": 2,
      "79101008030": 2,
      "79101008050": 2,
      "79101009010": 2,
      "79101999000": 2,
      "79102004010": 1,
      "79121001040": 3,
      "79121001042": 2,
      "79121004040": 2,
      "79121008040": 2
    }
  },
  "364513": {
    "product": "79101000090",
    "components": {
      "74996100010": 1,
      "79101001090": 1,
      "79101001091": 1,
      "79101001092": 1,
      "79101001093": 1,
      "79101001094": 1,
      "79101003010": 1,
      "79101003020": 1,
      "79101003030": 1,
      "79101003040": 1,
      "79101003050": 1,
      "79101003065": 1,
      "79101003070": 1,
      "79101003080": 1,
      "79101004010": 1,
      "79101004012": 1,
      "79101004020": 1,
      "79101004030": 1,
      "79101004031": 1,
      "79101004040": 1,
      "79101004042": 1,
      "79101005010": 1,
      "79101005020": 1,
      "79101005060": 1,
      "79101006010": 1,
      "79101007010": 1,
      "79101008010": 1,
      "79101008020": 1,
      "79101008030": 1,
      "79101009010": 1,
      "79101999000": 1
    }
  },
  "534201": {
    "product": "79101000041",
    "components": {
      "79101001048": 1,
      "79101003010": 1,
      "79101003020": 1,
      "79101003030": 1,
      "79101003040": 1,
      "79101003050": 1,
      "79101003065": 1,
      "79101003070": 1,
      "79101003080": 1,
      "79101004010": 1,
      "79101004012": 1,
      "79101004020": 1,
      "79101004030": 1,
      "79101004031": 1,
      "79101004040": 1,
      "79101004042": 1,
      "79101005010": 1,
      "79101005020": 1,
      "79101005060": 1,
      "79101006010": 1,
      "79101007010": 1,
      "79101008010": 1,
      "79101008020": 1,
      "79101008030": 1,
      "79101008050": 1,
      "79101009010": 1,
      "79101999000": 1,
      "79102004010": 1,
      "79121004040": 1,
      "79121008040": 1
    }
  }
}
```

Figure 9: Three Products selected by the logic

This has solved the first part of the business problem as the logic shows all product have a given list of subcomponents. The next question is how to determine the most likely finished product out of the selected products?

- **Created a New Feature (STNLR_MATNR):** This feature was derived from the concatenation of the STNLR (BOM Unique ID) and MATNR (Parent/Finished Product) features to allow us to derive a unique identifier for each bill of material and its corresponding finished product. This was also important to ensure accurate frequency of occurrence in subcomponents for each specific bill of material instance.

	STNLR_MATNR	IDNRK	IDNRK_count
0	364014_79101000040	79101003010	2
1	364014_79101000040	79101003020	2
2	364014_79101000040	79101003030	2
3	364014_79101000040	79101003040	1
4	364014_79101000040	79101003050	2
...
70	534201_79101000041	79101008010	1
71	534201_79101000041	79101008020	1
72	534201_79101000041	79101008030	1
73	534201_79101000041	79101009010	1
74	534201_79101000041	79101999000	1

Figure 10: New feature derived STNLR_MATNR

- **Data Transformation:** To ensure that the data better represent the underlying problem to the predictive model, the data was transformed. The subcomponents (IDNRK) became the features, their respective frequencies (IDNRK_count) as rows in the dataset with the new feature STNLR_MATNR becoming the target feature to be predicted. The test component list was transformed to this data structure. The data transformation steps were also implemented on the test component list set.
- **Dealing with categorical variables:** The STNLR_MATNR categorical variable representing the target feature to be predicted was modified by using label encoding to represent the different finished products. This becomes are training data for the machine learning model.

STNLR_MATNR		IDNRK_count													
IDNRK	79101003010	79101003020	79101003030	79101003040	79101003050	79101003065	79101003070	79101003080	79101004010	...	79101005010	79101005020	79101005060	79101006010	
0	0	2	2	2	1	2	1	2	2	2	...	2	2	2	2
1	1	1	1	1	1	1	1	1	1	1	...	1	1	1	1
2	2	1	1	1	1	1	1	1	1	1	...	1	1	1	1

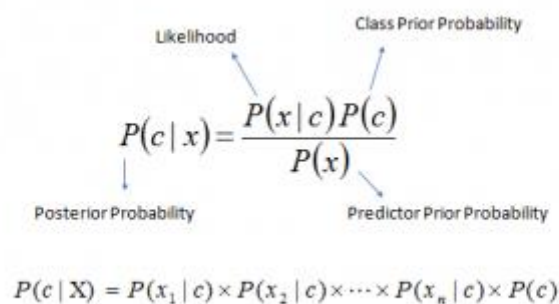
3 rows x 26 columns

Figure 11: Transformed data to feed into the predictive model

2.5 Model Selection

The next step was to decide which model would predict the most likely product. From our understanding, this is an optimization problem. Hence, we decided to go with the **Naïve Bayes Model**. A Naïve Bayes classifier is a probabilistic machine learning model that's used for classification task. The Bayes theorem lies at the core of the classifier.

Bayes Theorem:



The diagram shows the Bayes Theorem formula with labels pointing to its components. The formula is $P(c|x) = \frac{P(x|c)P(c)}{P(x)}$. Arrows point from the following labels to the corresponding parts of the formula: 'Likelihood' points to $P(x|c)$, 'Class Prior Probability' points to $P(c)$, 'Posterior Probability' points to $P(c|x)$, and 'Predictor Prior Probability' points to $P(x)$. Below the main formula, the joint probability formula is given: $P(c|X) = P(x_1|c) \times P(x_2|c) \times \dots \times P(x_n|c) \times P(c)$.

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

Labels in the diagram:

- Likelihood (points to $P(x|c)$)
- Class Prior Probability (points to $P(c)$)
- Posterior Probability (points to $P(c|x)$)
- Predictor Prior Probability (points to $P(x)$)

$$P(c|X) = P(x_1|c) \times P(x_2|c) \times \dots \times P(x_n|c) \times P(c)$$

Figure 12: Bayes Theorem Formula

$P(c/x)$ is the posterior probability of *class c* given *predictor (features)*.

$P(c)$ is the probability of *class*.

$P(x/c)$ is the likelihood which is the probability of *predictor* given *class*.

$P(x)$ is the prior probability of *predictor*.

We can calculate the probability of c occurring if x has already occurred using Bayes' theorem. The evidence is x , and the hypothesis is c . The predictors/features are assumed to be independent in this case. That is, the presence of one feature has does not affect the other. As a result, it is said to be naïve.

The Naive Bayes classifier has the advantage of being simple and quick to predict the test data set's class. It's also good at multi-class prediction. A Naive Bayes classifier performs well when the premise of independence stays true, and there is less training data. The Naive Bayes classifier is a good approach for this project as we a small training data.

3 Results

The Naive Bayes Classifier was trained using the training data and then deployed on the test data which is the given component list.

```
# Run Naive Bayes Model
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB()
clf.fit(dfm_NB.drop(['STNLR_MATNR'], axis=1), dfm_NB['STNLR_MATNR'])
```

```
# Run the Model on the test data
print ("Predicted Product : ", clf.predict(X))
print ("Product Prediction Probability Distribution : ", clf.predict_proba(X))
```

```
Predicted Product : [0]
Product Prediction Probability Distribution : [[0.33701651 0.33149174 0.33149174]]
```

Figure 13: Implementation of Naive Bayes Model

After the model was completed, the outcome of the probability distribution score between the three products were displayed. In this case, based on the higher probability (0.33701651), the predicted product can be assumed to be [0] i.e., STNLR is **364014_79101000040**.

4 Conclusion and Future Work

From the outcome of this project, the finished product in a dairy processing industry is identified from subcomponents contained in Bill of Material using Machine Learning. In this project, a supervised machine learning approach is performed with the implementation of the Naive Bayes classifier. The future work of this project can be done to improve the database which is used in training the model. Currently the training data is quite simple and small. A multi level BOM data could be incorporated and tested with the established machine learning pipeline. Also, with a larger training data sample, other machine learning algorithms could be tested and compared with the already existing solution.

Furthermore, it is important to test this solution in the infrastructure of the client as a consulting company. Hence, this solution should be tested on the client's environment and the effectiveness should be measured. This can be done by aligning the BOM Intelligence solution with the client's KPIs and observing the performance for a specific duration. In addition, this BOM Intelligence solution can be deployed into an interactive UI/UX platform for business stakeholders to easily access and make informed data-driven decisions.

5 References

- Browniee, J. (2014, September 26). *Discover Feature Engineering, How to Engineer Features and How to Get Good at It*. Machine Learning Mastery. <https://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/>
- Dear Systems. (2021, September 30). *The Importance of Bill of Materials in Manufacturing*. <https://dearsystems.com/the-importance-of-bill-of-materials-in-manufacturing/>
- Grant, M. (2022, March 13). *Bill of Materials (BOM)*. Investopedia. <https://www.investopedia.com/terms/b/bill-of-materials.asp>
- IHS Markit. (2016). *BOM Intelligence*. <https://cdn.ihs.com/www/pdf/BOM-Intelligence-2017.pdf>
- OpenBOM (openbom.com). (2018, February 9). *OpenBOM Part Catalogs and Bill of Material example*. <https://medium.com/@openbom/openbom-part-catalogs-and-bill-of-material-example-2ff29ad8b601>
- Zhou, C., & Cao, Q. (2019). Design and implementation of intelligent manufacturing project management system based on bill of material. *Cluster Computing*, 22, 8647–8655. <https://doi.org/10.1007/s10586-018-1934-4>