DATA ANALYTICS AND VISUALIZATION

ASSIGNMENT 2

MAY 2021

NAME: CHONG WEN HAO

STUDENT ID: 2000585

TO: DR LIEW HOW HUI

Background and Objective of the Study

The AI4I 2020 predictive maintenance dataset is a synthetic dataset that reflects real-world predictive maintenance data encountered in industry. It is provided by Stephan Matzka from School of Engineering, Berlin, Germany. The overall dataset contains 10,000 data points in rows and 14 features in columns. The product ID is unique for all product variants, starts with the H, M and L which represents the product is high, medium or low quality. The proportion of high product quality is 20%, medium product quality is 30% and low product quality is 50%. Besides, the air temperature and process temperature are generated using random walk process.

Furthermore, the rotation speed measured in rpm are calculated from a power of 2860 Watt., torque measured in Nm are normally distributed around 40Nm with no negative values. Tool wear measured in minute are 5/3/2 minutes to product quality variants H/M/L. The target feature in the maintenance prediction is the machine failures. If the machine has failed, it reflects on any of the failure modes such as tool wear failure (TWF), heat dissipation failure (HDF), power failure (PWF), overstrain failure (OSF) or random failures (RNF). The machine failure column has values of 1 or 0 only such that 1 indicates the machine has failed while 0 indicates machine has not failed.

The main objective of the study is to understand the relationships of these maintenance features, and each features' distribution. After that, using the machine learning models to classify the machine failure classes as accurate as possible. Lastly, the best maintenance predictive model will be selected to predict the machine failure classes.

Exploratory Data Analysis

Exploratory Data Analysis (EDA) is the first step to conduct after collecting a full datasets. It is important as it studies the dataset behaviors so that we can have better understandings on the data and the features. It can be conducted using simple statistics including mean, standard deviation, dispersion, and so on. It helps us to discover the patterns, anomalies, missing values, checking errors and interesting relations among the features.

There are several useful tables and charts that have been done using the python data analysis libraries such as numpy, scipy, pandas and etc.

1)

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 1 to 10000
Data columns (total 13 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Product ID               10000 non-null  object
 1   Type                     10000 non-null  category
 2   Air temperature [K]      10000 non-null  float64
 3   Process temperature [K]  10000 non-null  float64
 4   Rotational speed [rpm]   10000 non-null  int64
 5   Torque [Nm]              10000 non-null  float64
 6   Tool wear [min]          10000 non-null  int64
 7   Machine failure          10000 non-null  int8
 8   TWF                      10000 non-null  int8
 9   HDF                      10000 non-null  int8
 10  PWF                      10000 non-null  int8
 11  OSF                      10000 non-null  int8
 12  RNF                      10000 non-null  int8
dtypes: category(1), float64(3), int64(2), int8(6), object(1)
memory usage: 615.4+ KB
memory usage is reduced by 0.4MB
```

The table above is done using df.info( ). This code can shows 13 features' data types whether they are object, category, 8/16/32/64-bits integer or float. After converting the machine failure features from int32 to int8. the memory usage is greatly reduced by 400KB to 615.4KB. In addition, this table also shows that none of the features have missing values.

2)

	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]
count	10000.0000	10000.0000	10000.0000	10000.0000	10000.0000
mean	300.0049	310.0056	1538.7761	39.9869	107.9510
std	2.0003	1.4837	179.2841	9.9689	63.6541
min	295.3000	305.7000	1168.0000	3.8000	0.0000
25%	298.3000	308.8000	1423.0000	33.2000	53.0000
50%	300.1000	310.1000	1503.0000	40.1000	108.0000
75%	301.5000	311.1000	1612.0000	46.8000	162.0000
max	304.5000	313.8000	2886.0000	76.6000	253.0000

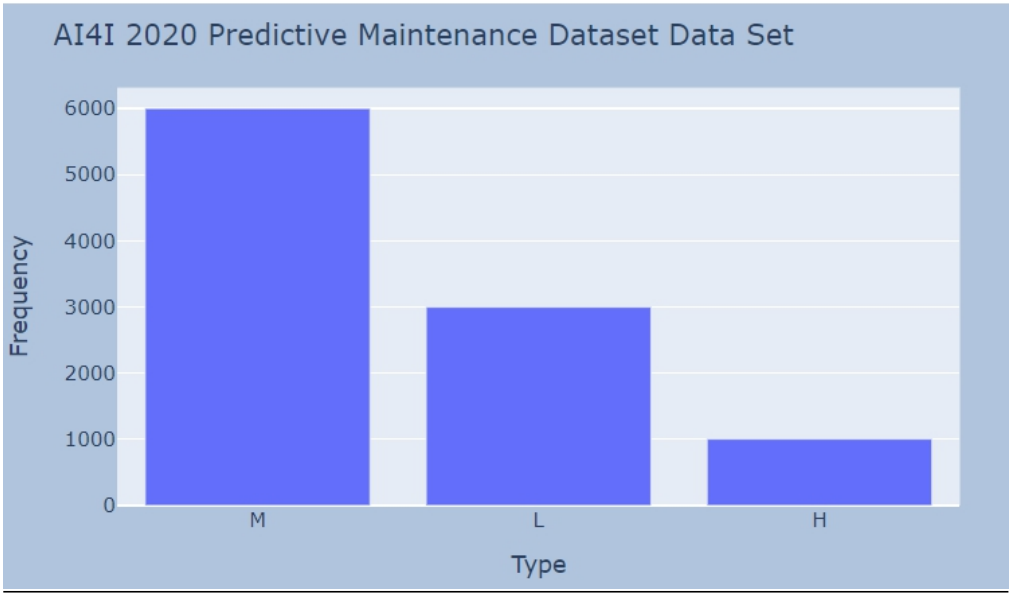
This table shows the frequency, mean, standard deviation, minimum value, first-quartile, median, third-quartile and the maximum value of the numerical data including air temperature, process temperature, rotational speed, torque and tool wear. Based on that, the average air temperature is 300K with low standard deviation of 2. The median value is same as its mean value which implies it is mostly normally distributed. The average process temperature is 310K which is 10K higher than average air temperature. Apart from that, the average rotational speed is 1539 rotation per minute and its standard deviation is 179. The maximum rotation speed is accounted for 2886 rpm which is faster than 99.5% of the observation while median value of 1503 rpm is lower than the average. Obviously, the distribution of rotation speed is right-skewed. The average torque length is 40 Nm with standard deviation of 10 Nm. Last, the average tool wear is 108 minutes with standard deviation of 63.7.

3)

Machinery	Machine failure		TWF		HDF		PWF		OSF		RNF	
isFailure	0	1	0	1	0	1	0	1	0	1	0	1
count	9661.00	339.00	9954.00	46.00	9885.00	115.00	9905.00	95.00	9902.00	98.00	9981.00	19.00
percentage(%)	96.61	3.39	99.54	0.46	98.85	1.15	99.05	0.95	99.02	0.98	99.81	0.19

This table shows the frequency and the percentage of the machine failures. The machine failure only occupies 3.39% while machine not failure occupies large proportion of 96.61% as overall. Hence, the dataset has imbalance class distribution whereby the machine failure is regarded as rare event.

4)



The bar chart shows the frequency of the product quality variants. Medium-quality product has 6000 quantities, followed by low-quality product which has 3000 quantities. The least type is high-quality product which has 1000 quantities.

```
The Machine Failure for High Product Quality is 2.0937%
The Machine Failure for Low Product Quality is 3.9167%
The Machine Failure for Medium Product Quality is 2.7694%
```

1) Pivot Tables

	Air temperature [K]		Process temperature [K]		Rotational speed [rpm]		Torque [Nm]		Tool wear [min]	
	mean	std	mean	std	mean	std	mean	std	mean	std
Machine failure										
0	299.9740	1.9907	309.9956	1.4868	1540.2600	167.3947	39.6297	9.4721	106.6937	62.9458
1	300.8864	2.0715	310.2903	1.3637	1496.4867	384.9435	50.1681	16.3745	143.7817	72.7599

	Air temperature [K]		Process temperature [K]		Rotational speed [rpm]		Torque [Nm]		Tool wear [min]	
	mean	std	mean	std	mean	std	mean	std	mean	std
TWF										
0	300.0036	2.0005	310.0048	1.4838	1538.6495	179.1547	39.9968	9.9669	107.4500	63.3666
1	300.2978	1.9460	310.1652	1.4840	1566.1739	205.7685	37.8370	10.2772	216.3696	12.2572

	Air temperature [K]		Process temperature [K]		Rotational speed [rpm]		Torque [Nm]		Tool wear [min]	
	mean	std	mean	std	mean	std	mean	std	mean	std
HDF										
0	299.9752	1.9916	309.9964	1.4883	1541.1205	178.9549	39.8336	9.9017	107.9598	63.6576
1	302.5609	0.6019	310.7887	0.6445	1337.2609	34.7460	53.1670	6.2235	107.1913	63.6293

	Air temperature [K]		Process temperature [K]		Rotational speed [rpm]		Torque [Nm]		Tool wear [min]	
	mean	std	mean	std	mean	std	mean	std	mean	std
PWF										
0	300.0043	1.9989	310.0060	1.4826	1536.6163	168.2316	39.9051	9.6342	108.0092	63.6479
1	300.0758	2.1471	309.9547	1.6004	1763.9684	620.8291	48.5147	26.7887	101.8842	64.3557

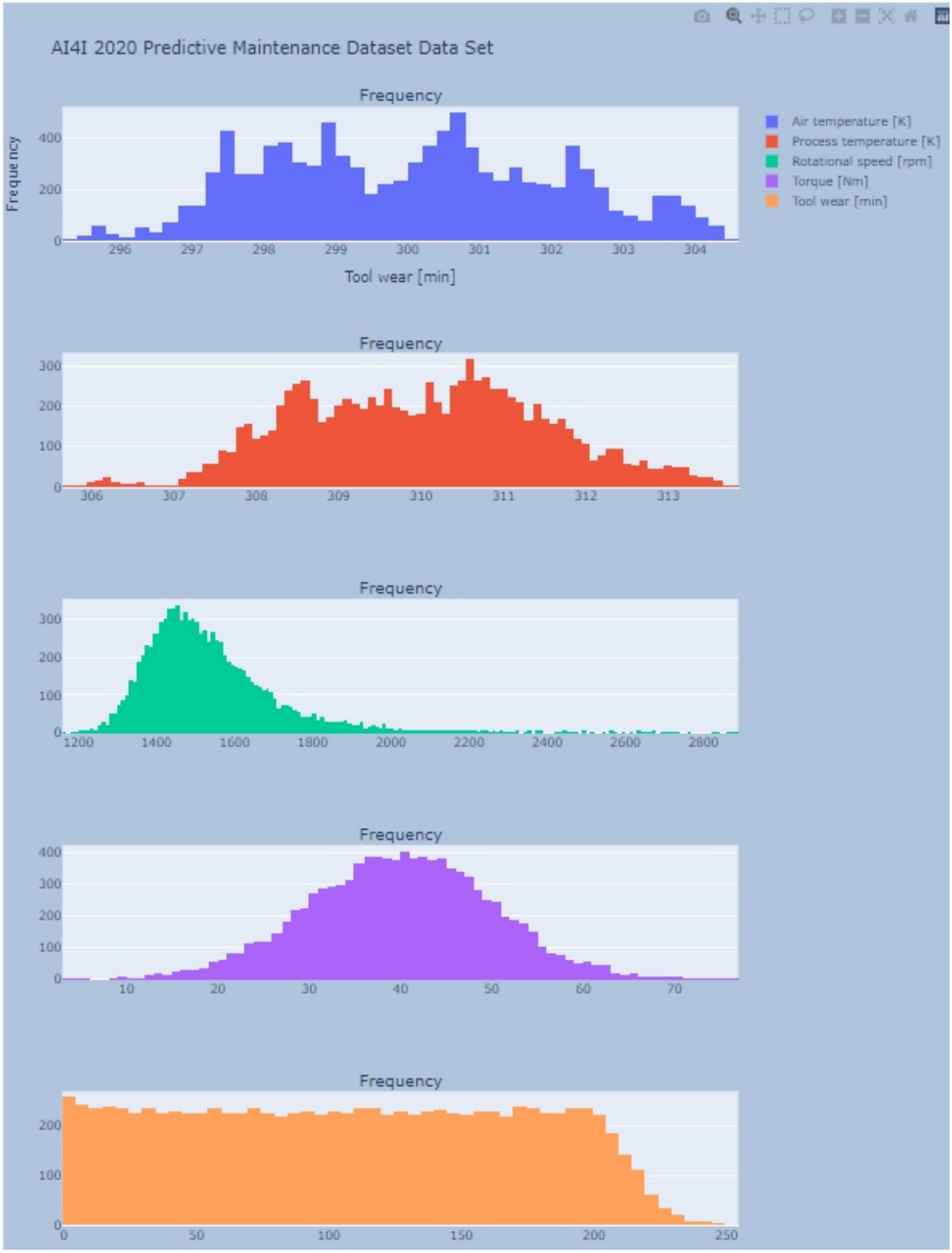
	Air temperature [K]		Process temperature [K]		Rotational speed [rpm]		Torque [Nm]		Tool wear [min]	
	mean	std	mean	std	mean	std	mean	std	mean	std
OSF										
0	300.0045	2.0001	310.0049	1.4835	1540.6412	179.0787	39.8050	9.8305	106.9638	63.1669
1	300.0449	2.0286	310.0735	1.5110	1350.3265	61.2508	58.3704	5.9436	207.6939	15.8110

	Air temperature [K]		Process temperature [K]		Rotational speed [rpm]		Torque [Nm]		Tool wear [min]	
	mean	std	mean	std	mean	std	mean	std	mean	std
RNF										
0	300.0034	2.0005	310.0041	1.4835	1538.8785	179.3791	39.9799	9.9670	107.9195	63.6421
1	300.8158	1.7076	310.7632	1.4595	1485.0000	109.4390	43.6737	10.5759	124.4737	69.5504

	Air temperature [K]		Process temperature [K]		Rotational speed [rpm]		Torque [Nm]		Tool wear [min]	
	mean	std	mean	std	mean	std	mean	std	mean	std
Type										
H	299.8670	2.0218	309.9257	1.4894	1538.1476	173.1334	39.8383	9.6423	107.4197	63.0801
L	300.0158	1.9875	310.0123	1.4752	1539.4692	180.4285	39.9966	10.0123	108.3788	64.0582
M	300.0293	2.0174	310.0188	1.4984	1537.5989	179.0598	40.0173	9.9922	107.2723	63.0446

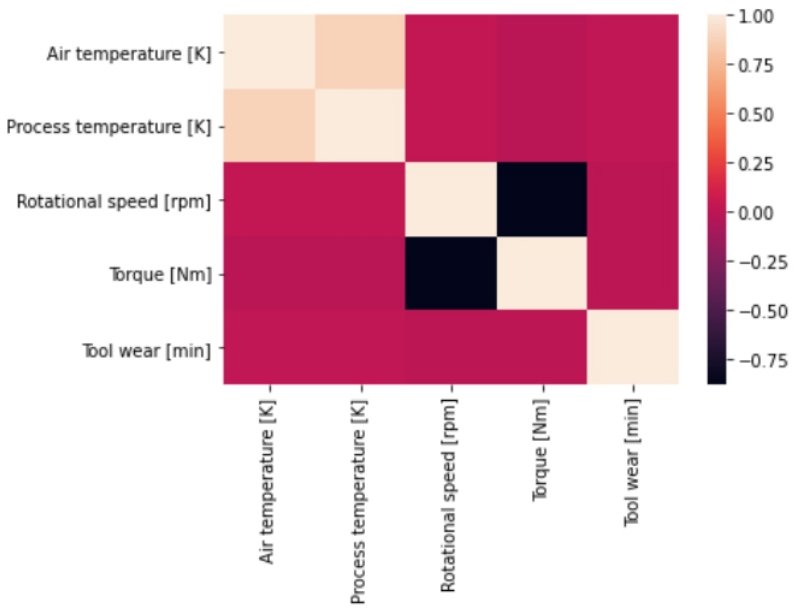
	Machine failure		TWF		HDF		PWF		OSF		RNF	
	mean	std	mean	std	mean	std	mean	std	mean	std	mean	std
Type												
H	0.0209	0.1432	0.0070	0.0833	0.0080	0.0890	0.0050	0.0705	0.0020	0.0446	0.0040	0.0631
L	0.0392	0.1940	0.0042	0.0644	0.0127	0.1118	0.0098	0.0987	0.0145	0.1195	0.0022	0.0465
M	0.0277	0.1641	0.0047	0.0682	0.0103	0.1012	0.0103	0.1012	0.0030	0.0547	0.0007	0.0258

5)



The above charts are the list of histograms using plotly, python. By plotting these, it clearly shows the distribution of the numerical features.

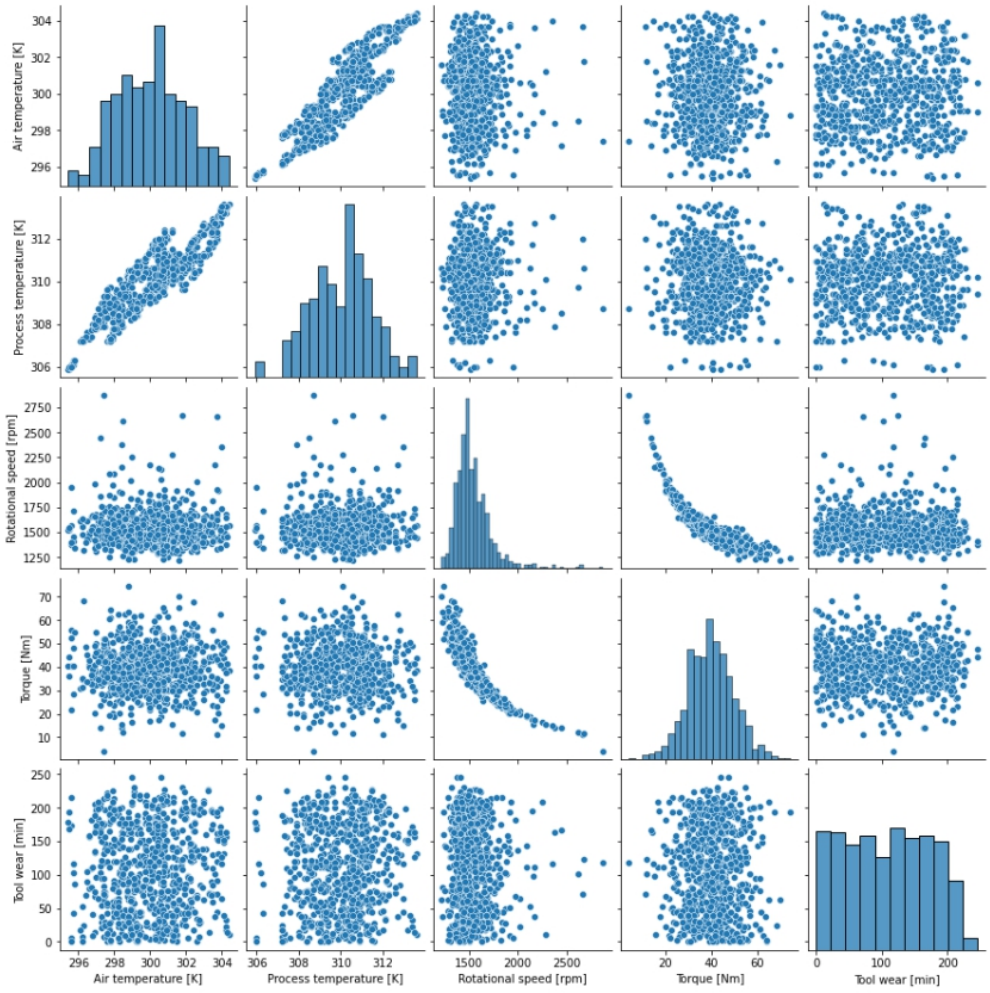
6)



	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]
Air temperature [K]	1.000000	0.876107	0.022670	-0.013778	0.013853
Process temperature [K]	0.876107	1.000000	0.019277	-0.014061	0.013488
Rotational speed [rpm]	0.022670	0.019277	1.000000	-0.875027	0.000223
Torque [Nm]	-0.013778	-0.014061	-0.875027	1.000000	-0.003093
Tool wear [min]	0.013853	0.013488	0.000223	-0.003093	1.000000

The above shows the heat map and correlation table for numerical features. The red colors indicate weak correlation or no correlation between two numerical features, the milk colors indicate significantly positive relationship between two numerical features while the black colors indicate significantly negative relationship between two numerical features. The interval of correlation value is between -1 and 1. Each individual features' correlation value is showing in the correlation table.

7)



The above shows the pair plots of 5 numerical features. The sample size of this pair plot is 750 observation which is randomly selected. The diagonal charts are the histogram and off-diagonal charts are the scatter plots. The scatter plot is useful to visualize if two numerical features are correlated or not. Two numerical features are not correlated when we do not find any patterns of the scatter plots for x and y.

Feature Engineering

Feature engineering is the process of transforming the raw data into features that better represent the underlying problem. This maintenance dataset does not have any missing values, and hence we do not need to withdraw or impute the missing values. However, there is one categorical feature which is not numerical data type should be transformed into numerical data type. One of the optimal ways is to use One-Hot Encoder to transform the machine quality variants (type) into binary data type. For example, H can encodes into [1, 0, 0], M encodes into [0, 1, 0] and L encodes into [0, 0, 1].

For the numerical features, the means and variances are different from the others. The features that have higher mean would have greater magnitude and dominate over other features in the predictions. Therefore, it is necessary to normalize or standardize the numerical features to achieve Gaussian with zero mean and unit variance. In this case, OneHotEncoder and StandardScaler from sklearn library are used to fit and transform the raw data into the features compatible to the machine learning models requirements. OneHotEncoder will transform the categorical feature into n-by-3 sparse array with binary values, and then transform into ordinary array type. Lastly, standard-scaled features are concatenated with the encoded features to get ready for the training data and testing data splits.

Furthermore, the characteristics of this maintenance dataset is that the values of target variable are only 1 or 0, whereby 0 accounted for nearly 97% while 1 accounted for just 3% out of total 10,000 observations. The class distribution is not equal or close to equal and hence can be considered as imbalanced dataset. The imbalanced dataset would cause several problems to the models. When the loss function is set to minimize the error / inaccuracy, there would be chances that the parameters are optimized towards higher accuracy such that all predictions give the value of machine not failure represented by 0, which can obtain 97% accuracy rate.

There are several ways to counter this imbalanced dataset including changing the evaluation metrics, over-sampling the minority, under-sampling the majority, and etc. In this case, over-sampling the minority is used through generating synthetic datasets on minority class using the Synthetic Minority Over-Sampling Techniques (SMOTE) from imbalance library, Python.

```
df = pd.read_csv("ai4i2020.csv", index_col = 0)
df.columns = ["product_id", "type", "air_temperature", "process_temperature",
              "rotational_speed", "torque", "tool_wear", "machine_failure",
              "TWF", "HDF", "PWF", "OSF", "RNF"]

X = df.drop(["product_id", "machine_failure", "TWF", "HDF", "PWF", "OSF", "RNF"], axis = 1).values
y = df.loc[:,"machine_failure"].values


x1 = X[:, 0] #categorical  << Encode
x2 = X[:, 1:] #number   << Scale

x1 = x1.reshape([-1, 1])
X_bin = OneHotEncoder().fit_transform(x1).toarray()
X_scale = StandardScaler().fit_transform(x2)

X = np.concatenate([X_bin, X_scale], axis = 1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 1)

oversample = SMOTE()
X_train, y_train = oversample.fit_resample(X_train, y_train)

Xy = np.concatenate([X_train, y_train.reshape([-1,1])], axis = 1)
np.random.shuffle(Xy)

X_train = Xy[:, :-1]
y_train = Xy[:, -1].reshape([-1,1])

y_test = y_test.reshape([-1, 1])
```

Predictive Modelling

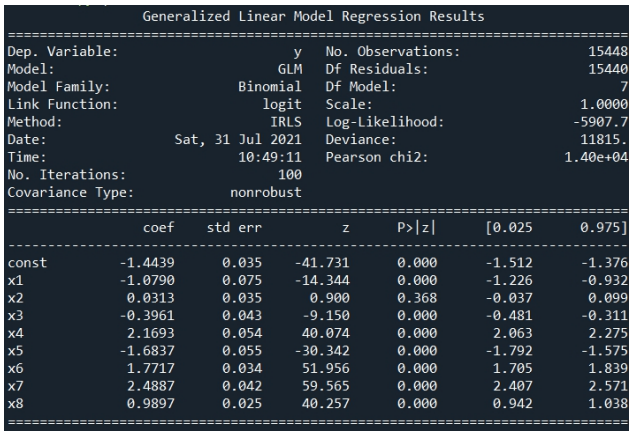
The predictive models used in this maintenance datasets will be machine learning models including logistic regression, support vector classifier, K-neighbors classifier, naive-bayes Gaussian NB, decision tree classifier, ensemble methods such as random forest classifier & extreme gradient boosting classifier, and artificial neural network. These machine learning models are effective in classifying the target classes or multi-classes. Some machine learning algorithms will classify the classes based on the probability such that 0 if the output is less than 0.5 and 1 if the output is more than 0.5. The training datasets will be used to train the models and optimize the parameters' values through minimizing the cost function. Lastly, the model performance will be evaluated through confusion matrix, classification report and area under curve (AUC) score. In addition, accuracy metrics will not be used in this case because the imbalance testing set will make the result become unreliable as compared with confusion matrix and recalls from classification report.

The testing datasets remain imbalanced since it is not used to train and optimize the models. Instead, it is used to test the models . Instead, it is used to test the models performance for the classification problems. It is not suggested to use the accuracy score to validate the performance because it does not provide accurate results for the model performance. Hence, confusion matrix and the indicators such as recalls in the classification report can be used to evaluate the model performance using the testing datasets.

1) Logistic Regression Model

In python, there are two libraries provide the logistic regression algorithm including statsmodels.GLM and sklearn.linear_model.LogisticRegression. The below will display the python code using the GLM statistical method.

```
X_train = sm.add_constant(X_train)

model = sm.GLM(y_train, X_train, sm.families.family.Binomial()).fit()
print(model.summary())

beta = model.params

y_proba_ = model.predict(sm.add_constant(X_test))


y_pred = sm.add_constant(X_test) @ beta

y_proba = 1/ (1+ np.exp(-y_pred)) # Sigmoid Function

print(np.sum(y_proba == y_proba_))

y_pred_bin = []

for i in y_proba:
    if i > 0.5:
        y_pred_bin.append(1)
    else:
        y_pred_bin.append(0)

y_pred_bin = np.array(y_pred_bin)

print(accuracy_score(y_test, y_pred_bin))
print(confusion_matrix(y_test, y_pred_bin))
print(classification_report(y_test, y_pred_bin))
```



Generalized Linear Model Regression Results

Dep. Variable:	y	No. Observations:	15448
Model:	GLM	Df Residuals:	15440
Model Family:	Binomial	Df Model:	7
Link Function:	logit	Scale:	1.0000
Method:	IRLS	Log-Likelihood:	-5907.7
Date:	Sat, 31 Jul 2021	Deviance:	11815.
Time:	10:49:11	Pearson chi2:	1.40e+04
No. Iterations:	100		
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-1.4439	0.035	-41.731	0.000	-1.512	-1.376
x1	-1.0790	0.075	-14.344	0.000	-1.226	-0.932
x2	0.0313	0.035	0.900	0.368	-0.037	0.099
x3	-0.3961	0.043	-9.150	0.000	-0.481	-0.311
x4	2.1693	0.054	40.074	0.000	2.063	2.275
x5	-1.6837	0.055	-30.342	0.000	-1.792	-1.575
x6	1.7717	0.034	51.956	0.000	1.705	1.839
x7	2.4887	0.042	59.565	0.000	2.407	2.571
x8	0.9897	0.025	40.257	0.000	0.942	1.038

```
[[1591  346]
 [   8   55]]
              precision    recall  f1-score   support

           0       0.99      0.82      0.90      1937
           1       0.14      0.87      0.24        63

    accuracy                           0.82      2000
   macro avg       0.57      0.85      0.57      2000
weighted avg       0.97      0.82      0.88      2000
```

Most of the parameters are statistically significant except rotational speed's parameter based on the p-value, Log-likelihood shows how fit the model to the datasets which is usually used to compare with other models. Accuracy, precision score and F1-score are not reliable to imbalance testing dataset as there are 1937 machine not failure class and only 63 machine failure. The amount of false positive will mostly be greater than true negative due to 96.85: 3.15 ratio. Therefore, the recall score will be the main concern to measure how well the model predict machine failure class and machine not

failure class correctly. Based on the above output of classification report, the recall for class 0 is 0.82 and class 1 is 0.87. Hence, the model correctly classifies slightly more than 4 out of 5 class 0 observation and 87 out of 100 class 1 observation.

The below shows the Logistic Regression using Logistic Regression.

```
def logisticRegression(self, query, predict = None):
    model = LogisticRegression(C=0.05179474679231213, penalty= "l2").fit(self.X_train, self.y_train)
    y_pred = model.predict(self.X_test)
```

```
In [5]: model.logisticRegression("confusion matrix")
Out[5]:
array([[1933,    0],
       [  68,    0]], dtype=int64)
```

The above hyperparmeters of the logistic regression are tuned using the GridSearchCV with scoring focused on recall-evaluation metrics. The confusion matrix shows that the total 2000 predictions on the testing dataset are class 0. The accuracy score is 96.6% which is not reliable. Instead, the recall score shows 100% accuracy on class 0 but 0% accuracy on class 1. Hence, the imbalanced datasets rise a problem that the model will tend to treat the rare event class 1 as noise and all predictions go to class 0.

```
array([[1583,  354],
       [   8,   55]], dtype=int64)
```

	precision	recall	f1-score	support
0	0.99	0.82	0.90	1937
1	0.13	0.87	0.23	63
accuracy			0.82	2000
macro avg	0.56	0.85	0.57	2000
weighted avg	0.97	0.82	0.88	2000

After using the balanced training datasets, the result of confusion matrix and classification report are similar to the logistic regression using generalized linear model method.

2) Support Vector Classifier (SVC)

```
def supportVectorClassifier(self, query, predict = None):
    model = SVC(C= 1000, gamma= 0.001, kernel= 'rbf').fit(X_train, y_train)
    y_pred = model.predict(X_test)
```

```
array([[1731,  206],
       [   4,   59]], dtype=int64)
```

	precision	recall	f1-score	support
0	1.00	0.89	0.94	1937
1	0.22	0.94	0.36	63
accuracy			0.90	2000
macro avg	0.61	0.92	0.65	2000
weighted avg	0.97	0.90	0.92	2000

Based on the result of recall score, SVC model correctly classifies 89 out of 100 class 0 observation and 94 out of 100 class 1 observation.

3) K-Neighbors Classifier

```
def kNeighborsClassifier(self, query, predict = None):
    model = KNeighborsClassifier(n_neighbors=4).fit(self.X_train, self.y_train)
    y_pred = model.predict(self.X_test)
```

```
array([[1855,   82],
       [  14,   49]], dtype=int64)
```

	precision	recall	f1-score	support
0	0.99	0.96	0.97	1937
1	0.37	0.78	0.51	63
accuracy			0.95	2000
macro avg	0.68	0.87	0.74	2000
weighted avg	0.97	0.95	0.96	2000

Based on the result of recall score, K-Neighbors Classifier model correctly classifies 96 out of 100 class 0 observation and 78 out of 100 class 1 observation.

4) Gaussian Naive Bayes (GaussianNB)

```
def naiveBayes(self, query, predict = None):
    model = GaussianNB(var_smoothing=1.0).fit(self.X_train, self.y_train)
    y_pred = model.predict(self.X_test)
```

```
array([[1863,   74],
       [  21,   42]], dtype=int64)
```

	precision	recall	f1-score	support
0	0.99	0.96	0.98	1937
1	0.36	0.67	0.47	63
accuracy			0.95	2000
macro avg	0.68	0.81	0.72	2000
weighted avg	0.97	0.95	0.96	2000

Based on the result of recall score, GaussianNB model correctly classifies 96 out of 100 class 0 observation and 67 out of 100 class 1 observation.

5) Decision Tree Classifier

```
def decisionTree(self, query, predict = None):
    model = DecisionTreeClassifier(min_samples_leaf = 3, max_features = 6, max_depth = None, criterion = 'entropy').
    y_pred = model.predict(self.X_test)
```

```
array([[1871,   66],
       [  13,   50]], dtype=int64)
```

	precision	recall	f1-score	support
0	0.99	0.96	0.98	1937
1	0.42	0.84	0.56	63
accuracy			0.96	2000
macro avg	0.71	0.90	0.77	2000
weighted avg	0.98	0.96	0.97	2000

Based on the result of recall score, Decision Tree Classifier model correctly classifies 96 out of 100 class 0 observation and 84 out of 100 class 1 observation.

6) Random Forest Classifier

```
def randomForest(self, query, predict = None):
    model = RandomForestClassifier(n_estimators = 100, max_features = 3, max_depth = 100, bootstrap = True).
    y_pred = model.predict(self.X_test)
```

```
array([[1905,   32],
       [   9,   54]], dtype=int64)
```

	precision	recall	f1-score	support
0	0.99	0.98	0.99	1937
1	0.60	0.84	0.70	63
accuracy			0.98	2000
macro avg	0.80	0.91	0.84	2000
weighted avg	0.98	0.98	0.98	2000

Based on the result of recall score, Random Forest Classifier model correctly classifies 98 out of 100 class 0 observation and 86 out of 100 class 1 observation.

7) Extreme Gradient Boosting Method

```
def extremeGradientBoosting(self, query, predict = None):
    model = XGBClassifier(objective = "binary:logistic", n_estimators = 100, seed = 123, verbosity = 0).
    y_pred = model.predict(self.X_test)
```

```
array([[1908,   29],
       [  10,   53]], dtype=int64)
```

	precision	recall	f1-score	support
0	0.99	0.99	0.99	1937
1	0.65	0.84	0.73	63
accuracy			0.98	2000
macro avg	0.82	0.91	0.86	2000
weighted avg	0.98	0.98	0.98	2000

Based on the result of recall score, Extreme Gradient Boosting Classifier model correctly classifies 98 out of 100 class 0 observation and 86 out of 100 class 1 observation.

8) Multi-layers Neural Network

```
Epoch 0: the training accuracy is 0.5
         the testing accuracy is 0.03150000050663948

Epoch 30: the training accuracy is 0.8724106550216675
          the testing accuracy is 0.7724999785423279

Epoch 60: the training accuracy is 0.9107328057289124
          the testing accuracy is 0.9539999961853027

Epoch 900: the training accuracy is 0.9542983174324036
           the testing accuracy is 0.9424999952316284

Epoch 930: the training accuracy is 0.9553340077400208
           the testing accuracy is 0.9200000166893005

Epoch 960: the training accuracy is 0.9319005608558655
           the testing accuracy is 0.871999979019165

Epoch 990: the training accuracy is 0.9439409375190735
           the testing accuracy is 0.8889999985694885
```

```
array([[1824,  113],
       [   9,   54]], dtype=int64)
```

	precision	recall	f1-score	support
0	1.00	0.94	0.97	1937
1	0.32	0.86	0.47	63
accuracy			0.94	2000
macro avg	0.66	0.90	0.72	2000
weighted avg	0.97	0.94	0.95	2000

Based on the result of recall score, Multi-layers Neural Network model correctly classifies 94 out of 100 class 0 observation and 86 out of 100 class 1 observation.

Hyperparameter Model Tuning

```
def GridSearchCV_(model, X, y):
    modelName = {"decisionTree": DecisionTreeClassifier(),
                 "randomForest": RandomForestClassifier(),
                 "extremeGradientBoosting": XGBClassifier(),
                 "logisticRegression": LogisticRegression(),
                 "supportVectorClassifier": SVC(),
                 "kNeighborsClassifier": KNeighborsClassifier(),
                 "naiveBayes": GaussianNB()}
    params = {"supportVectorClassifier": [{"C": [1, 10, 100, 1000], "kernel": ["linear"]},
                {"C": [1, 10, 100, 1000], "gamma": [0.001, 0.0001], "kernel": ["rbf"]}],
         "logisticRegression": {"C": np.logspace(-5, 8, 15), "penalty": ["l1", "l2"]},
          "decisionTree": {"max_depth": [3, None], "max_features": np.random.randint(1,9),
              "min_samples_leaf": np.random.randint(1, 9),"criterion": ["gini", "entropy"]},
          "kNeighborsClassifier": {"n_neighbors": list(range(1, 5))},
        "randomForest": {"bootstrap": [True], "max_depth": [80, 90, 100, 110], "max_features": [2, 3],
                "min_samples_leaf": [3, 4, 5], "min_samples_split": [8, 10, 12], "n_estimators": [100, 200, 300, 1000]},
          "extremeGradientBoosting": {"min_child_weight": [1, 5, 10], "gamma": [0.5, 1, 1.5, 2, 5], "subsample": [0.6, 0.8, 1.0],
                "colsample_bytree": [0.6, 0.8, 1.0], "max_depth": [3, 4, 5]},
          "naiveBayes": {'var_smoothing': np.logspace(0,-9, num=100)}}

    modelName_ = modelName_[model]
    params_ = params[model]
    gscv = GridSearchCV(params_, param_grid = modelName_, scoring = "recall", cv = 5).fit(X, y)
    return gscv.best_params_
```

GridSearchCV and RandomizedSearchCV from sklearn are used to tune the models' hyperparameters. Given a set of hyperparameters' values, each is tested with other hyperparameters to find the best combinations that can maximize the recall scores. The RandomizedSearchCV is used for decision tree classifiers and random forest classifiers as it would reduce the time to find the optimal solutions as compared with GridSearchCV for the decision tree based models.

Voting Ensemble

```
model = model(X_train, y_train, X_test, y_test)
y_pred1 = model.extremeGradientBoosting("y_pred")
y_pred2 = model.randomForest("y_pred")
y_pred3 = model.supportVectorClassifier("y_pred")

y_final = []

for i, j, k in list(zip(y_pred1, y_pred2, y_pred3)):
    total = i+j+k
    if total > 1:
        y_final.append(1)
    else:
        y_final.append(0)
y_final = np.array(y_final)
```

```
array([[1896,   41],
       [   8,   55]], dtype=int64)
```

	precision	recall	f1-score	support
0	1.00	0.98	0.99	1937
1	0.57	0.87	0.69	63
accuracy			0.98	2000
macro avg	0.78	0.93	0.84	2000
weighted avg	0.98	0.98	0.98	2000

Based on the result of recall score, the voting ensemble method correctly classifies 98 out of 100 class 0 observation and 87 out of 100 class 1 observation. It is by far the best performance for the unseen datasets.

Summary

```
(tensorflow) PS C:\Users\cwhwe\Desktop\May_2021\Data_Analytics\Assignment\A2_Code> python input_output.py
Machine Type: L
Air Temperature [k]: 299.1
Process Temperature [k]: 309.4
Rotation Speed [rpm]: 1439
Torque [Nm]: 44.2
Tool Wear [min]: 82
Result: machine not failure
```

```
(tensorflow) PS C:\Users\cwhwe\Desktop\May_2021\Data_Analytics\Assignment\A2_Code> python input_output.py
Machine Type: L
Air Temperature [k]: 298.9
Process Temperature [k]: 309.1
Rotation Speed [rpm]: 2861
Torque [Nm]: 4.6
Tool Wear [min]: 143
Result: machine failure
```

As a summary, the voting ensemble method formed by extreme gradient boosting method, random forest classifier and support vector classifier do not only provide high performance in the training datasets, but also provide the best performance to predict the chance of machine failure based on the features designed in the new datasets compared to other machine learning models. Lastly, it is recommended that the future study should consider choosing the suitable sampling techniques, transforming into the features that better predict the target including min-max scaler and developing more sophisticated models for the maintenance predictive datasets.