Data Science CSCI 3320 Project: Predictive Maintenance

Zaki Kurdya

About the domain

About the domain

Predictive maintenance refers to the use of data-driven, proactive maintenance methods that are designed to analyze the condition of equipment and help predict when maintenance should be performed.

About the domain

AI based predictive maintenance uses a variety of data from:

- IoT sensors imbedded in equipment's.
- Manufacturing operations.
- Environmental data.
- And more.

About the domain

AI models can:

- Look for patterns in data that indicate failure modes for specific components.
- Generate more accurate predictions of the lifespan for a component given environmental conditions.

About the domain

Predictive maintenance insights are an extremely valuable asset in improving the overall maintenance and reliability of an operation. Benefits include:

- Minimize the number of unexpected breakdowns.
- Maximize asset uptime and improve asset reliability.
- Maximize production hours.
- Improve safety.

About the domain

When predictive maintenance is working effectively as a maintenance strategy, maintenance is only performed on machines when it is required. This brings several cost savings:

- Minimizing the time the equipment is being maintained.
- Minimizing the production hours lost to maintenance.
- Minimizing the cost of spare parts and supplies.

About the domain

Predictive maintenance programs have also been shown to lead to a tenfold increase in ROI (Return on Investment) by:

- 25% - 30% reduction in maintenance costs.
- 70% - 75% decrease of breakdowns.
- 35% - 45% reduction in downtime.

About the domain

Predictive maintenance	Preventive maintenance
• Is proactive maintenance. • Uses predictive maintenance technology to address potential problems and schedule corrective maintenance before a failure occurs. • Does not often require machine downtime, and if it does, it’s generally short.	• Is planned maintenance, usually for set times and dates or after a specific data metric is reached. • Often utilizes scheduling software to notify teams or individuals of upcoming equipment maintenance. • Often requires machine downtime.

About the data

About the data

This synthetic dataset is modeled after an existing **milling machine.**

About the data

The dataset consists of **10,000 data points** stored as rows with **14 features** in columns:

1. **UID:** unique identifier ranging from 1 to 10000.
2. **product ID:** consisting of a letter L, M, or H for low (50% of all products), medium (30%) and high (20%) as product quality variants and a variant-specific serial number.
3. **type:** just the product type L, M or H from column 2.

About the data

4. **air temperature [kelvin]:** generated using a random walk process later normalized to a standard deviation of 2 K around 300 K.
5. **process temperature [kelvin]:** generated using a random walk process normalized to a standard deviation of 1 K, added to the air temperature plus 10 K.
6. **rotational speed [revolutions per minute]:** calculated from a power of 2860 W, overlaid with a normally distributed noise.
7. **torque [newton-meter]:** torque values are normally distributed around 40 Nm with a SD = 10 Nm and no negative values.

About the data

8. **tool wear [minutes]:** (breakdown and gradual failure of a cutting tool due to regular operation) The quality variants H/M/L add 5/3/2 minutes of tool wear to the used tool in the process.
9. **machine failure:** label that indicates, whether the machine has failed in this particular datapoint for any of the following failure modes are true. The machine failure consists of five independent failure modes.

About the data

Machine failure modes:

- Tool wear failure (TWF)
- Heat dissipation failure (HDF)
- Power failure (PWF)
- Overstrain failure (OSF)
- Random failures (RNF)

If at least one of the above failure modes is true, the process fails and the '**machine failure**' label is set to 1.

About the data

This dataset is part of the following publication:

S. Matzka, "Explainable Artificial Intelligence for Predictive Maintenance Applications," 2020 Third International Conference on Artificial Intelligence for Industries (AI4I), 2020, pp. 69-74, doi: 10.1109/AI4I49448.2020.00023.

Dataset link on UCI.

Data Exploration

Data exploration

- Dataset shape:

Number of rows: 10000, Number of columns: 14

UDI	Product ID	Type	Air temp. [K]	Process temp. [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	TWF	HDF	PWF	OSF	RNF
1	M14860	M	298.1	308.6	1551	42.8	0	0	0	0	0	0	0
2	L47181	L	298.2	308.7	1408	46.3	3	0	0	0	0	0	0
3	L47182	L	298.1	308.5	1498	49.4	5	0	0	0	0	0	0

Data exploration

- The dataset does not contain missing values

#	Column	Non-Null Count	Dtype
0	Type	10000 non-null	object
1	Air temperature	10000 non-null	float64
2	Process temperature	10000 non-null	float64
3	Rotational speed	10000 non-null	int64
4	Torque	10000 non-null	float64
5	Tool wear	10000 non-null	int64
6	Machine failure	10000 non-null	int64
7	TWF	10000 non-null	int64
8	HDF	10000 non-null	int64
9	PWF	10000 non-null	int64
10	OSF	10000 non-null	int64
11	RNF	10000 non-null	int64

Data exploration

Descriptive information on numerical attributes

index	count	mean	std	min	25%	50%	75%	max
Air temperature	10000.0	300.0049	2.0002	295.3	298.3	300.1	301.5	304.5
Process temperature	10000.0	310.0056	1.4837	305.7	308.8	310.1	311.1	313.8
Rotational speed	10000.0	1538.7761	179.2841	1168.0	1423.0	1503.0	1612.0	2886.0
Torque	10000.0	39.9869	9.9689	3.8	33.2	40.1	46.8	76.6
Tool wear	10000.0	107.951	63.6541	0.0	53.0	108.0	162.0	253.0

Data exploration

Descriptive information on numerical attributes

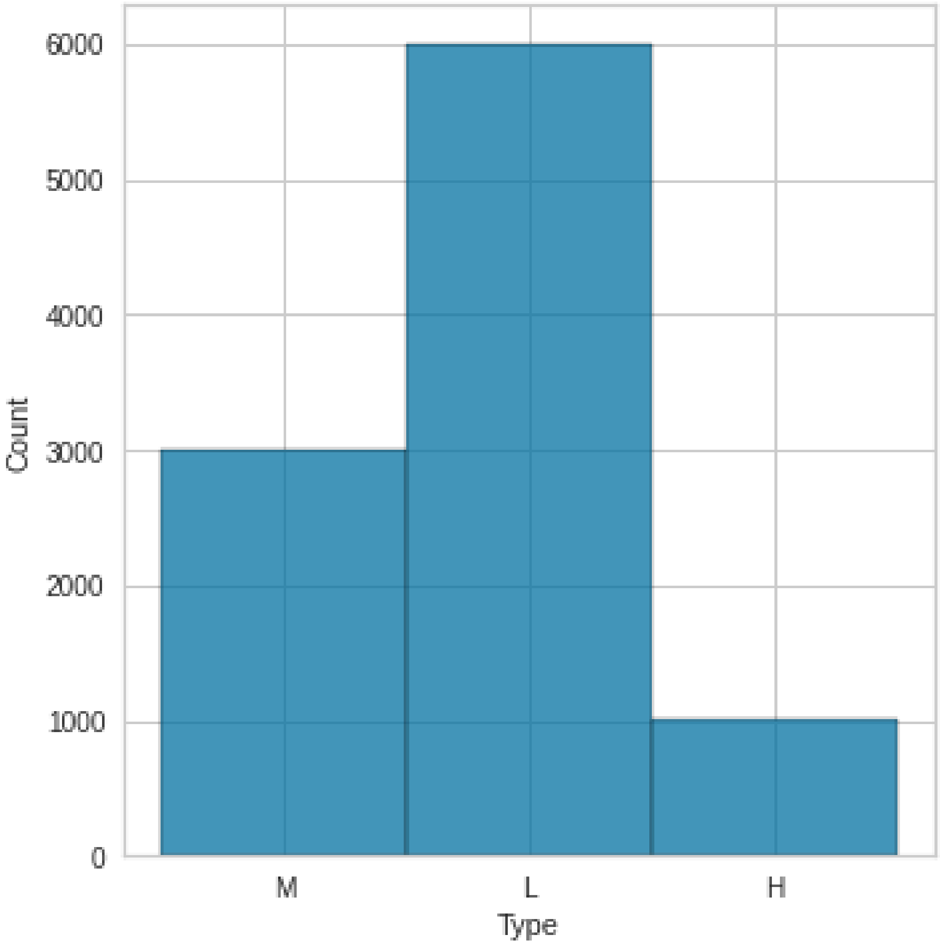
index	count	mean	std	min	25%	50%	75%	max
Machine failure	10000.0	0.0339	0.1809	0.0	0.0	0.0	0.0	1.0
TWF	10000.0	0.0046	0.0676	0.0	0.0	0.0	0.0	1.0
HDF	10000.0	0.0115	0.1066	0.0	0.0	0.0	0.0	1.0
PWF	10000.0	0.0095	0.0970	0.0	0.0	0.0	0.0	1.0
OSF	10000.0	0.0098	0.0985	0.0	0.0	0.0	0.0	1.0

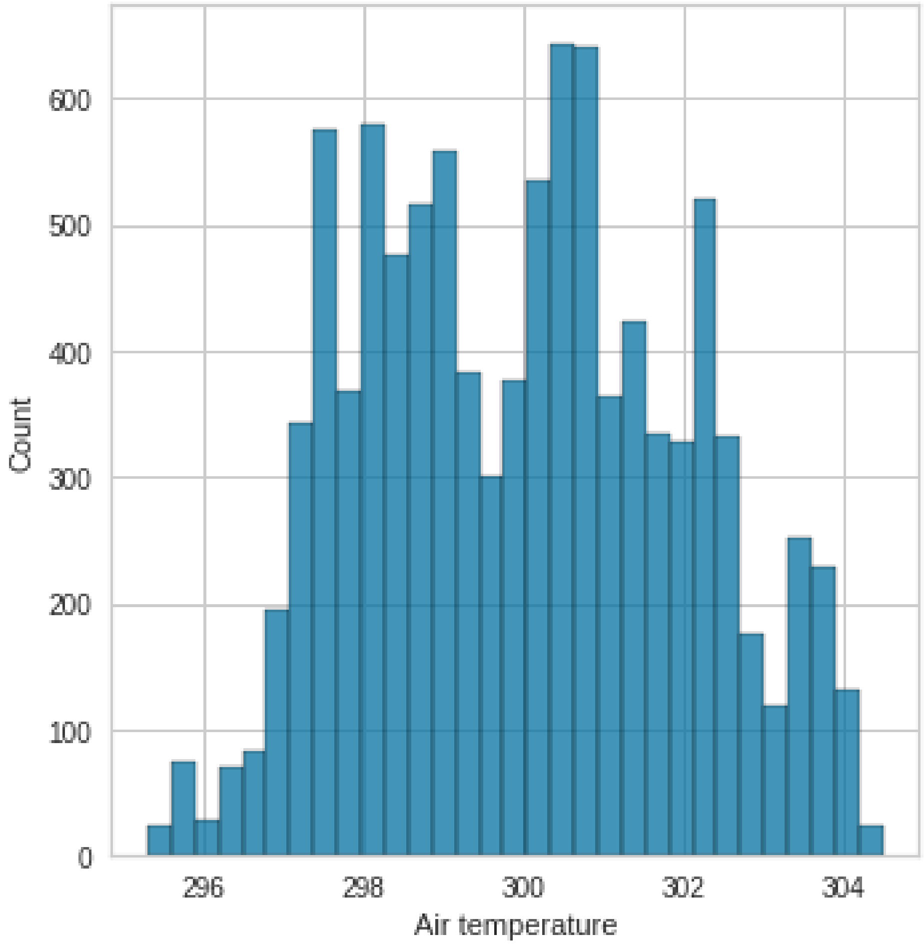
Data exploration

Descriptive information on categorical attributes

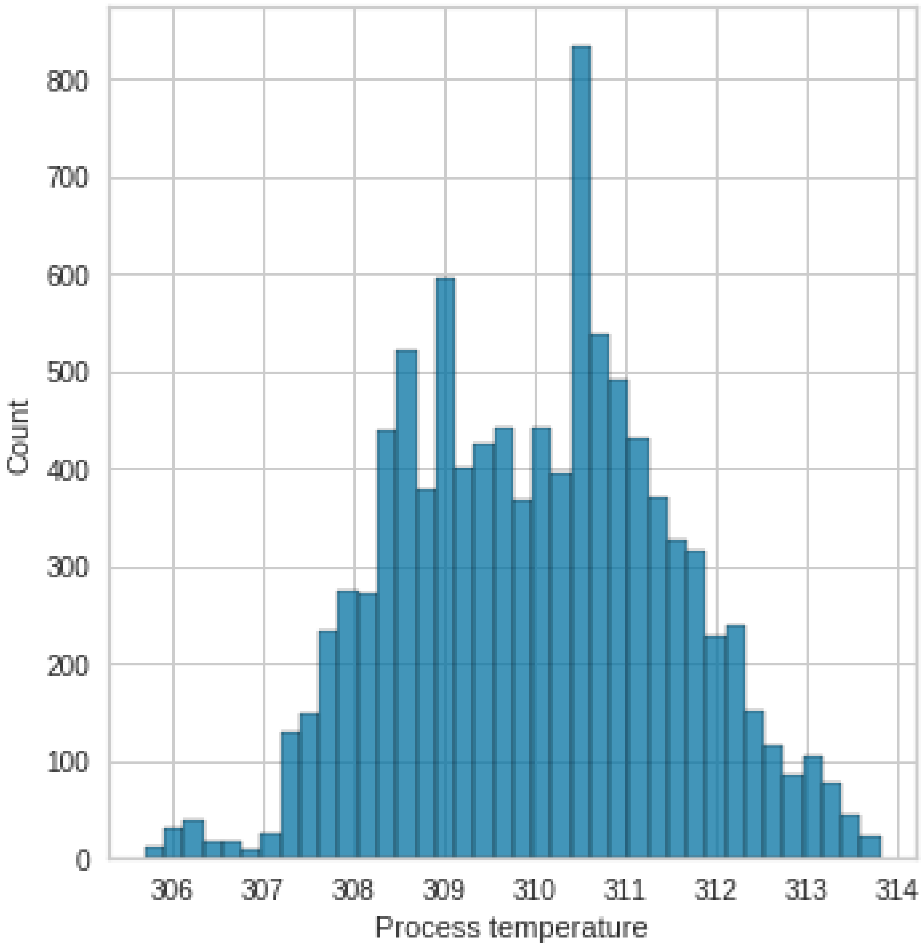
index	count	unique	top	frequency
Type	10000	3	L	6000

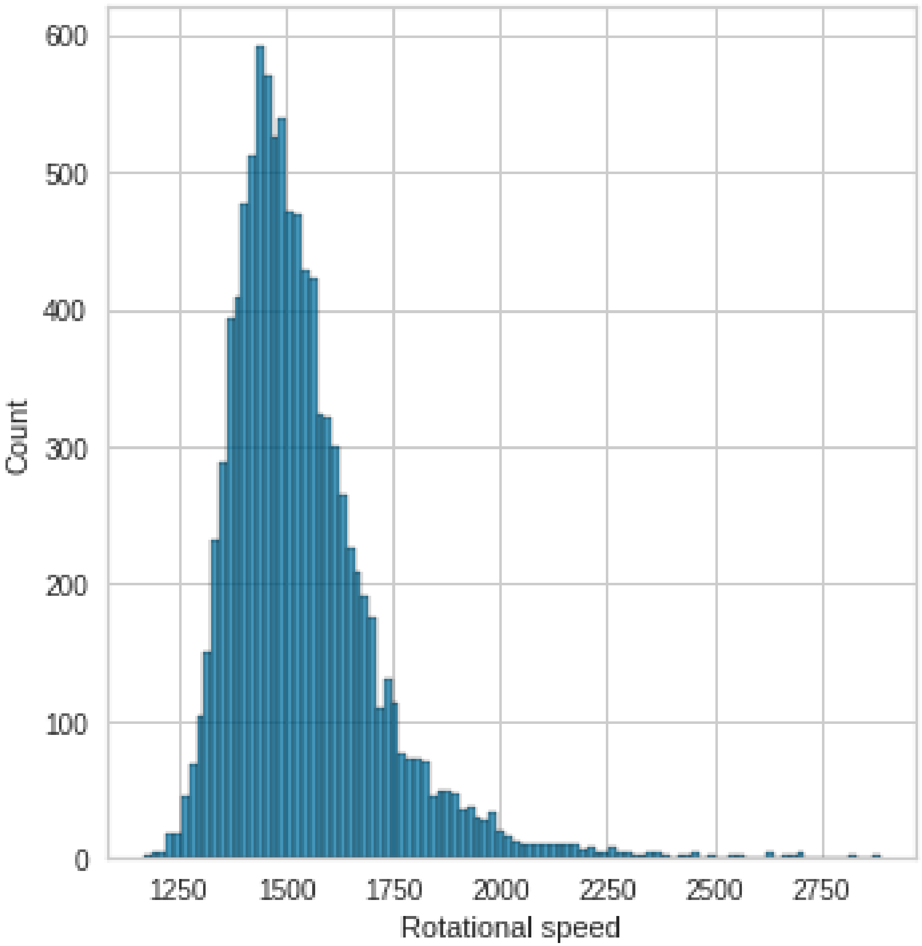
Data exploration - visualizations



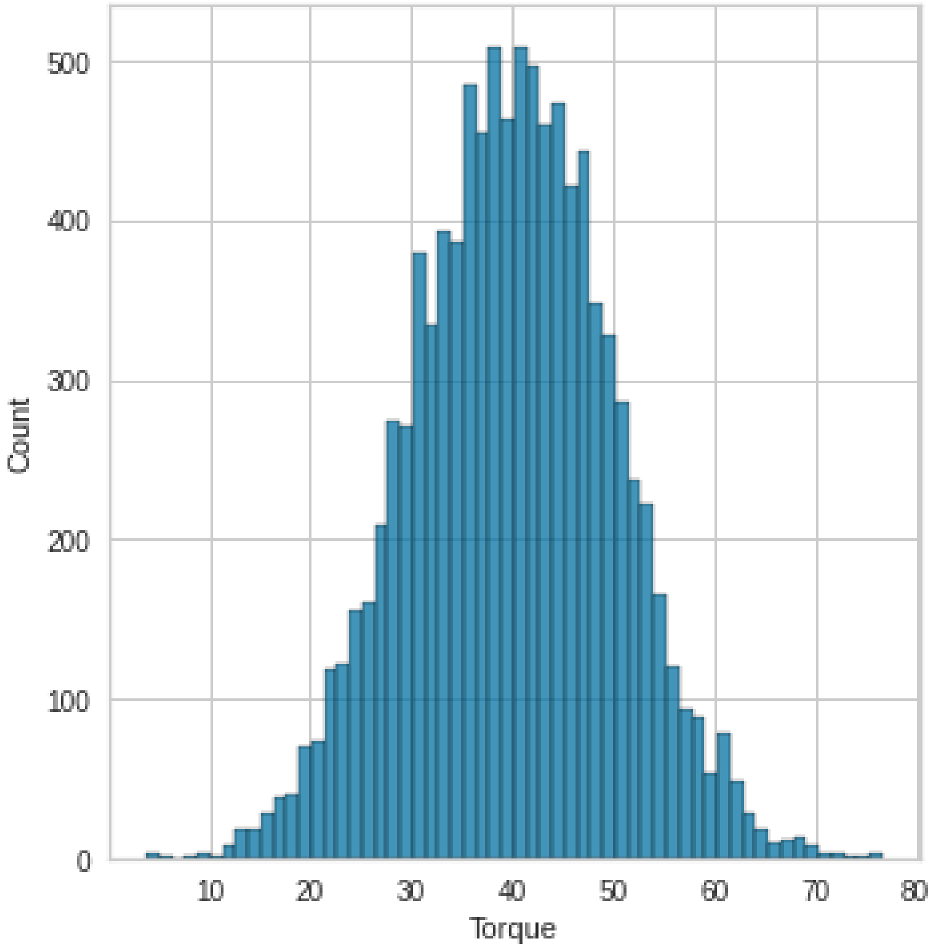


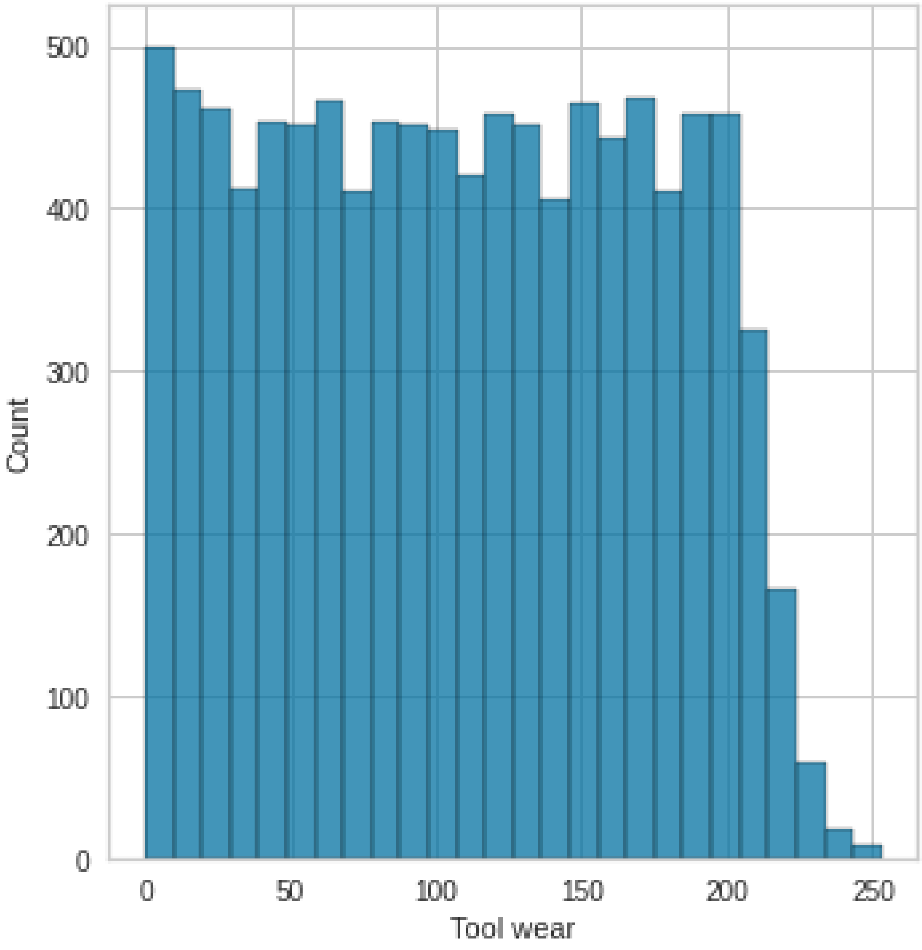
Data exploration - visualizations



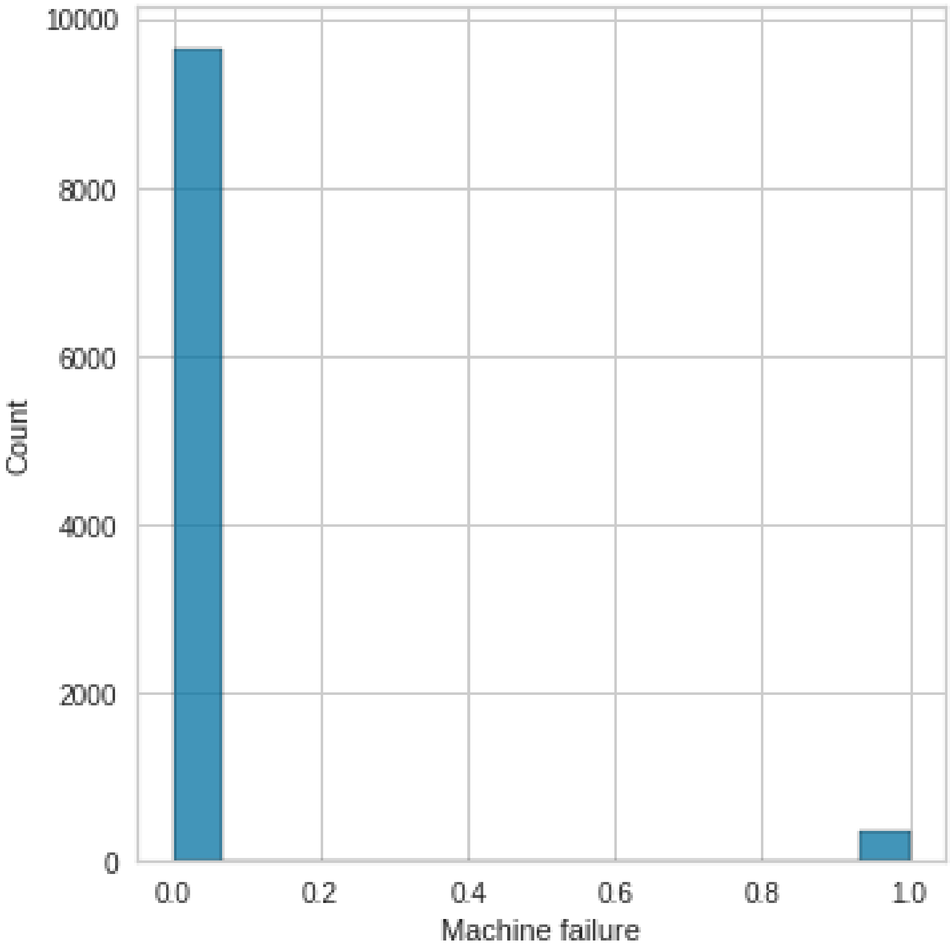


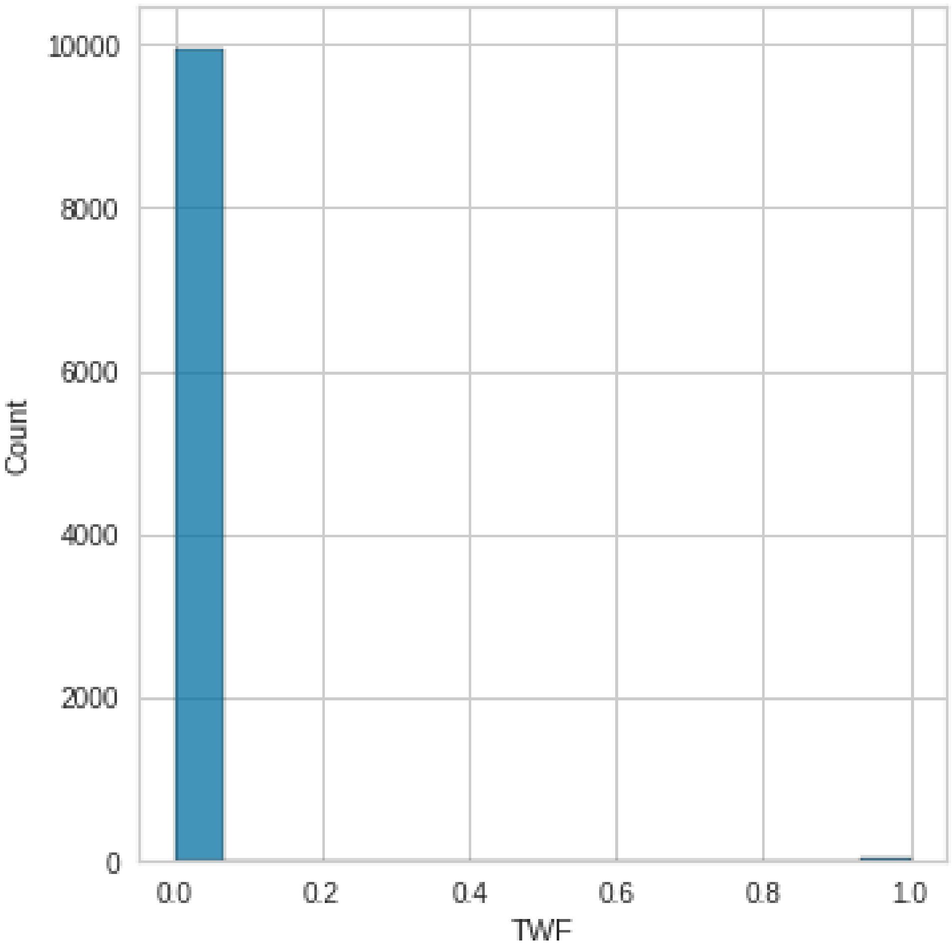
Data exploration - visualizations



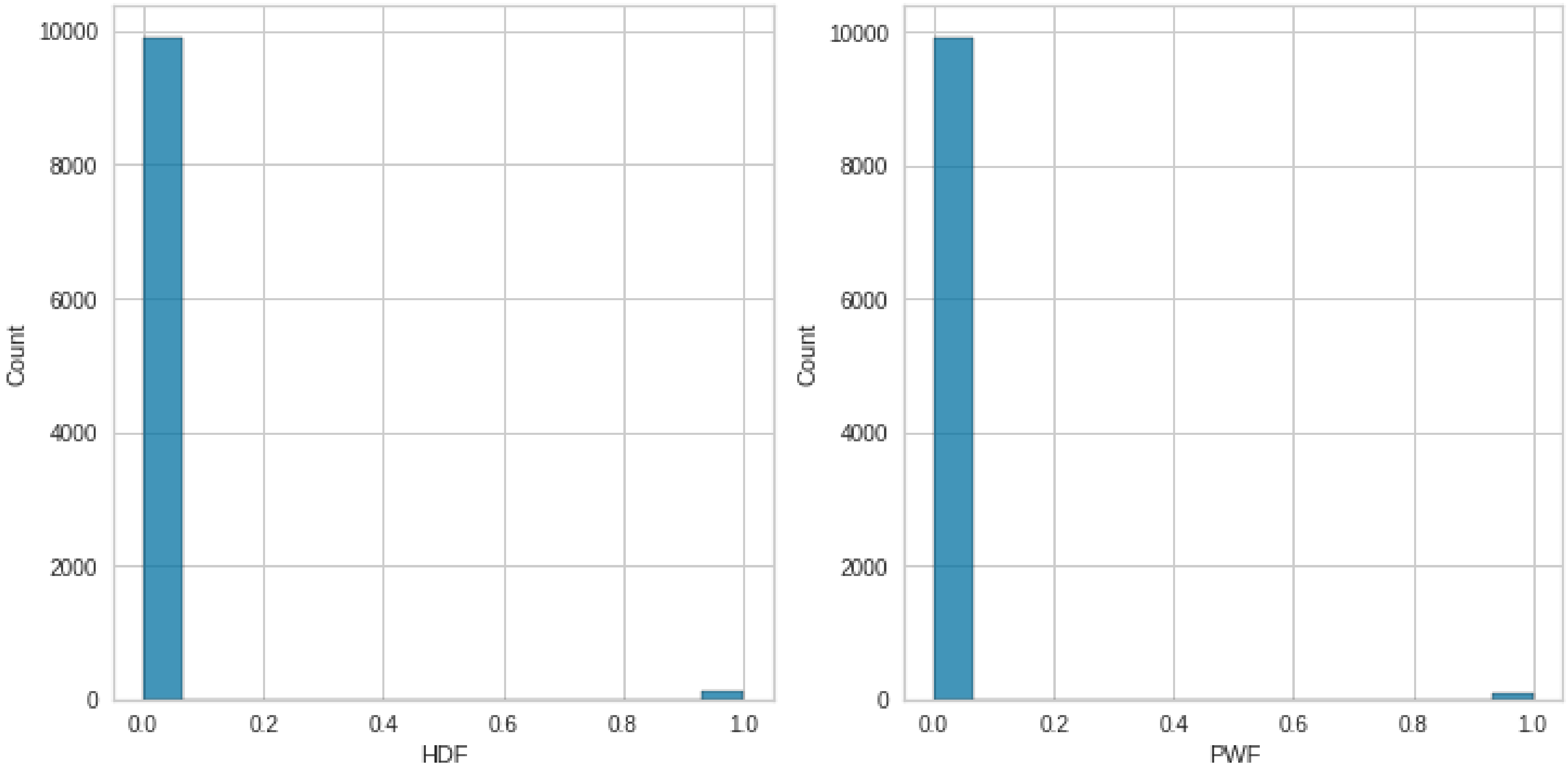


Data exploration - visualizations

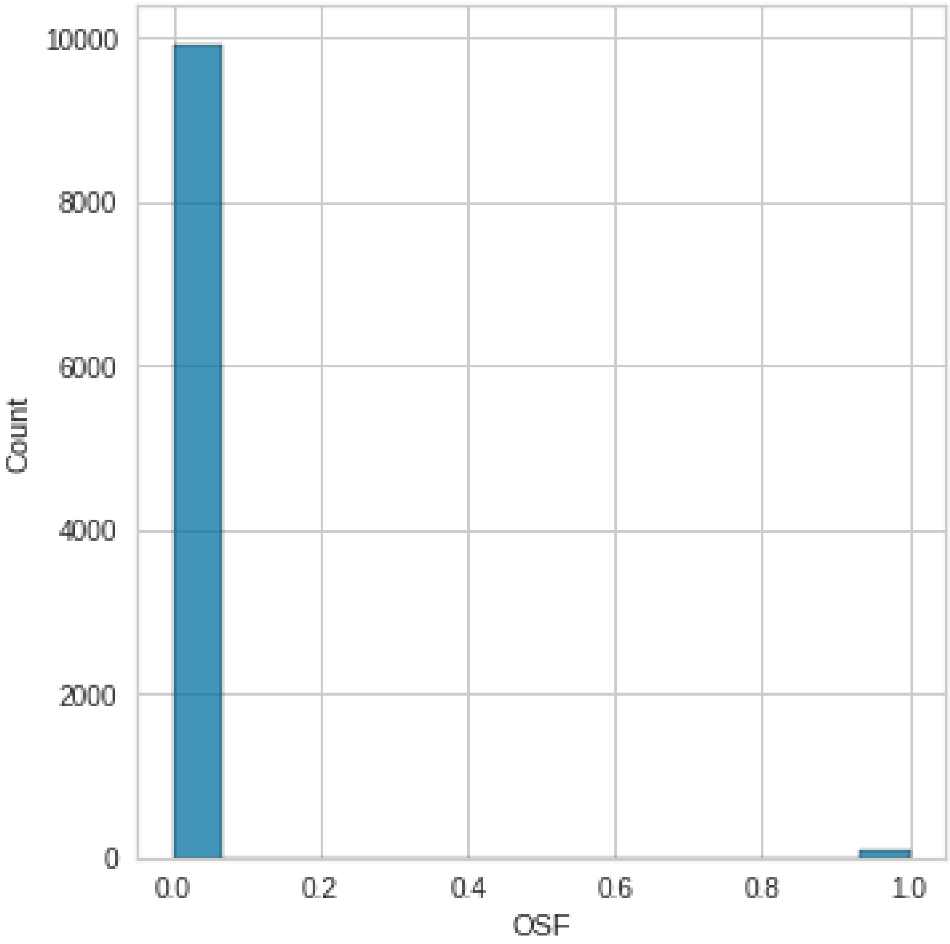


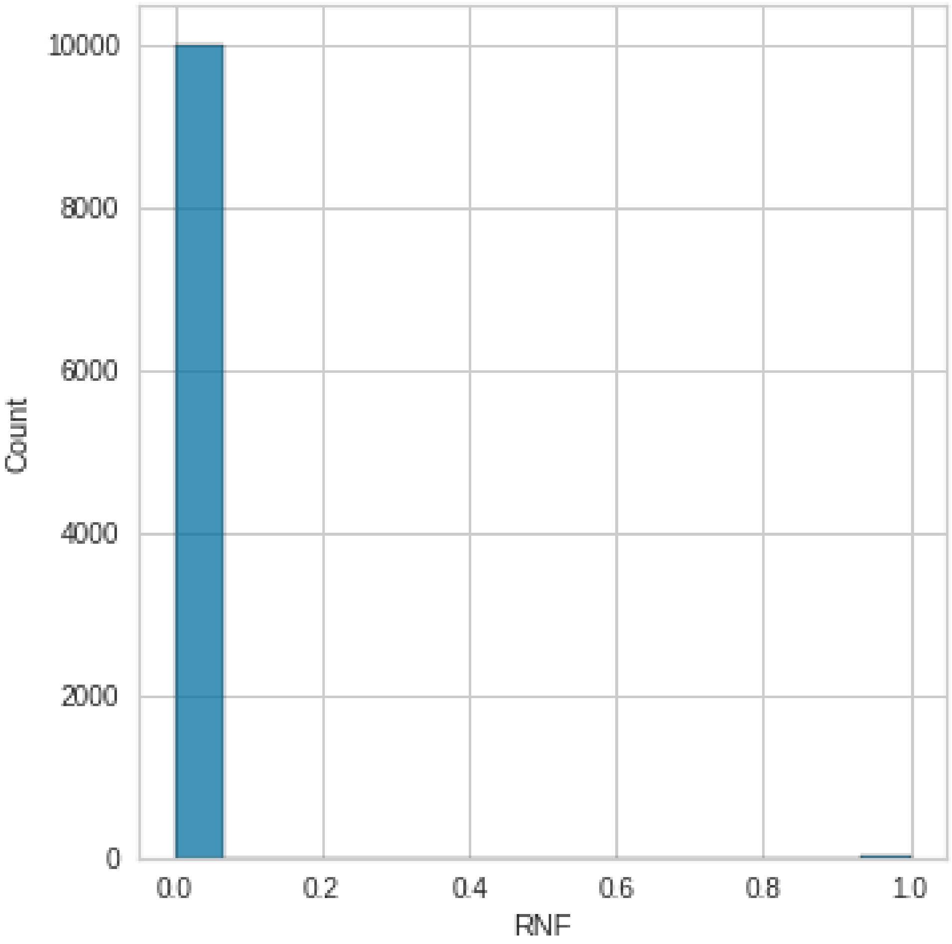


Data exploration - visualizations



Data exploration - visualizations





Data exploration - visualizations

We can see that the data is **imbalanced** on these attributes:

- Type
- Machine failure
- TWF, HDF, PWF, OSF, RNF

Data exploration - visualizations

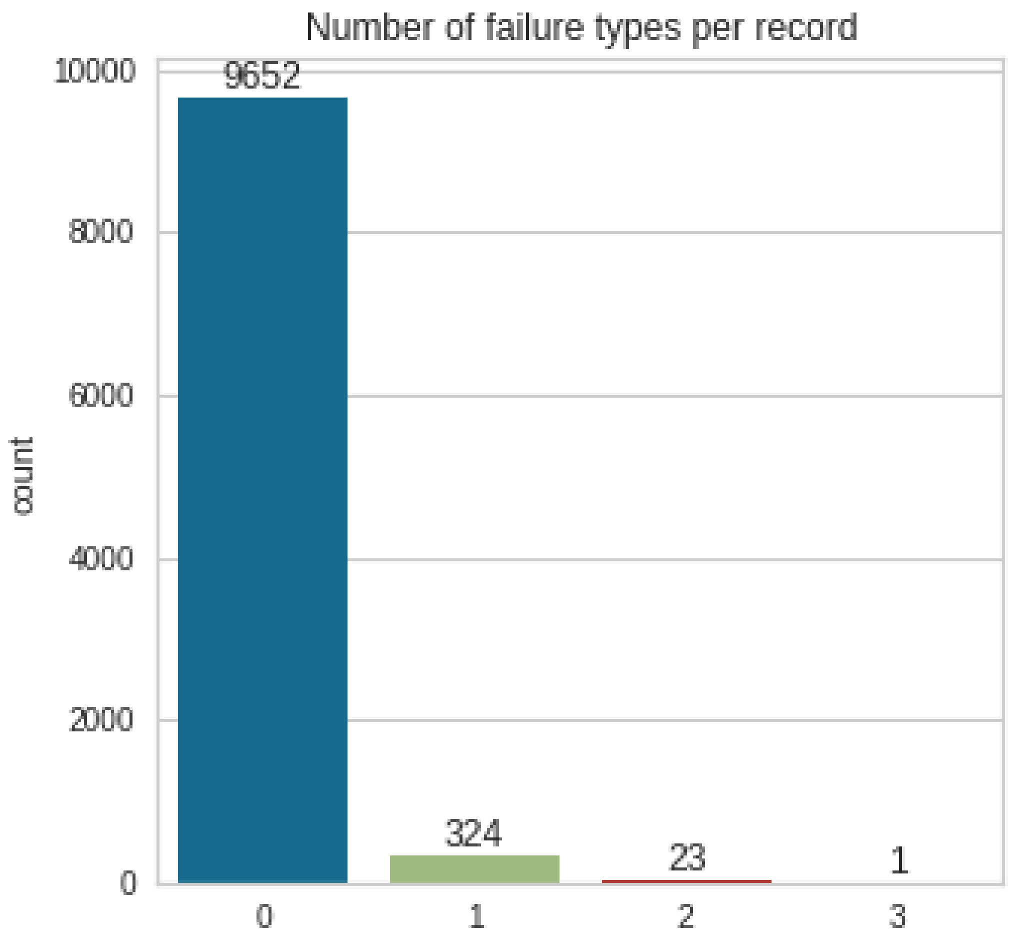
We can see that the data is **normally distributed** across these attributes:

- Air temperature
- Process temperature
- Rotational speed (skewed right)
- Torque

Data exploration - visualizations

We can see that the data is roughly **uniformly distributed** on (Tool wear).

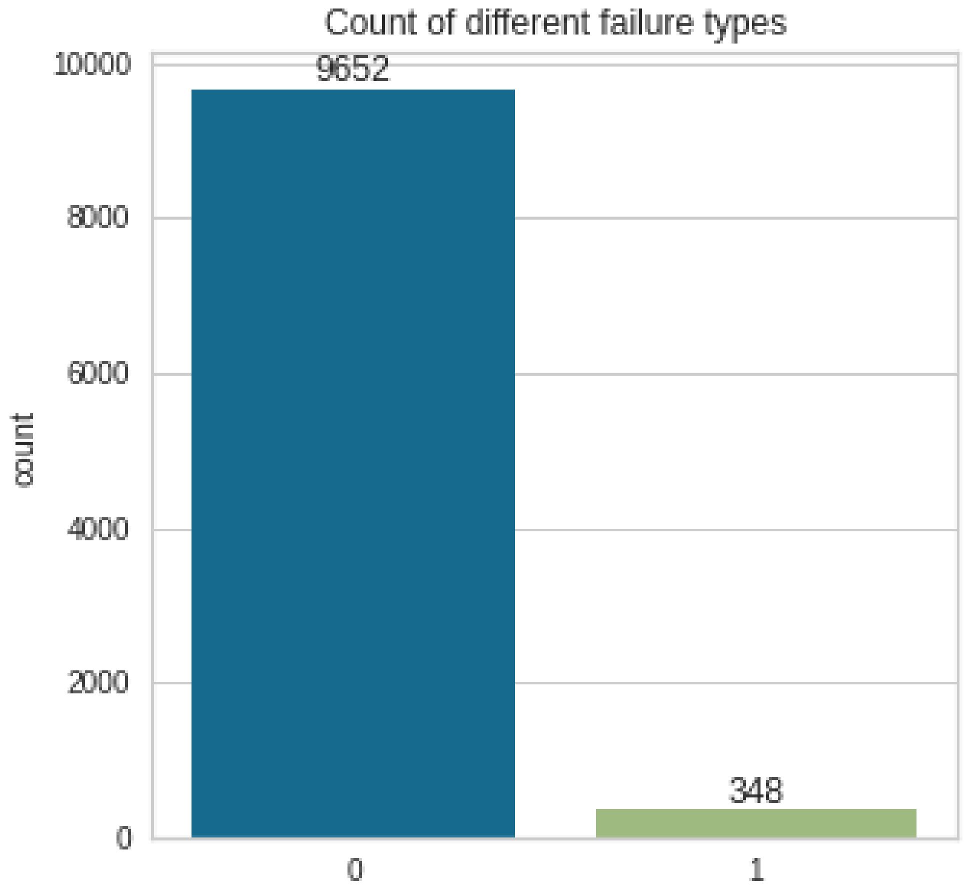
Data exploration - visualizations



As shown here, 24 records contain more than one type of failure, but their count is very small compared to the entire dataset, so we will combine the failure types into one feature, and then drop the individual ones.

Data exploration - visualizations

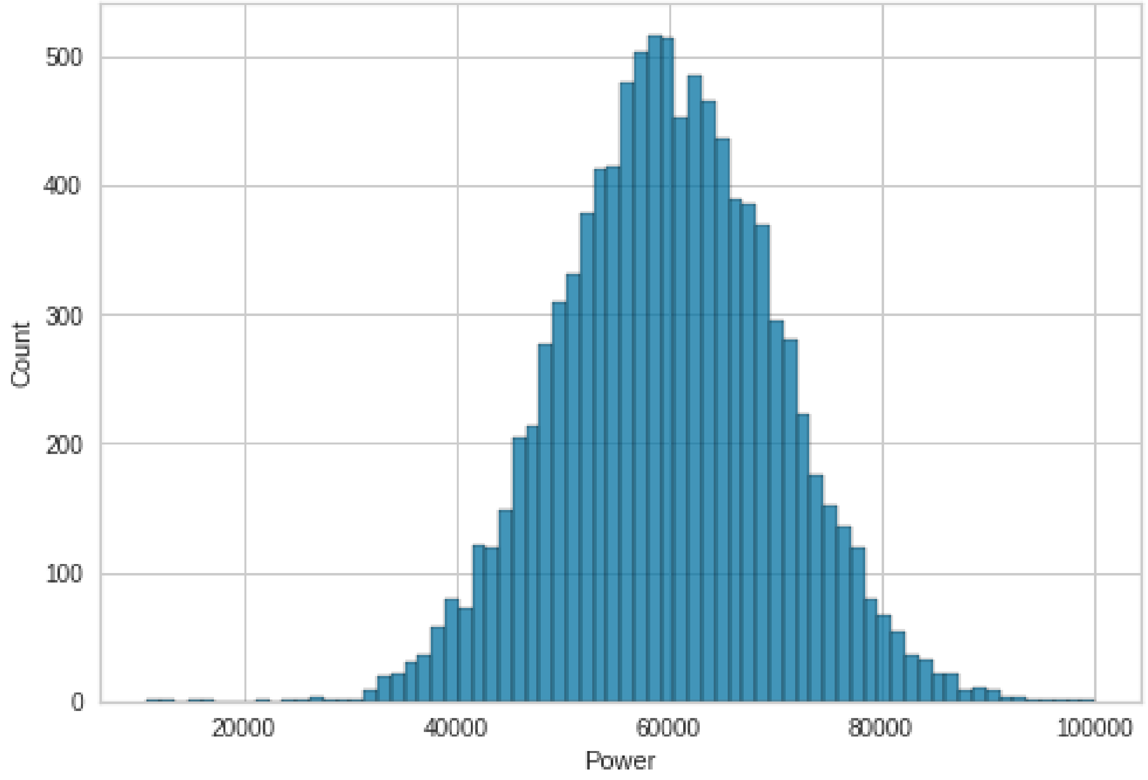
The result after the last edit (it's still biased, so we'll try to oversample it before training the ML models).



Data exploration

We can derive a new attribute (Power) using this formula:

$$Power = Torque \times Rotational\ speed$$



Data Preparation

Data preparation

Data type conversion:

- First, convert **Type** attribute into numbers, such that:
  L = 0, M = 1, and H = 2.
- Then convert each attribute to float for easier processing later.

Data preparation

Handling outliers:

Calculate and handle the outliers for each attribute using **IQR** (interquartile range) and **LOF** (Density-Based Anomaly Detection).

Number of rows after removing outliers: **9400**

Data preparation

IQR (interquartile range):

```
for col in df.columns:
    if col not in excluded_columns:
      # calculate the IQR (interquartile range)
      Q1 = df[col].quantile(0.25)
      Q3 = df[col].quantile(0.75)
      IQR = Q3 - Q1
      outliers = df[(df[col] <= (Q1 - 1.5 * IQR)) | (df[col] >= (Q3 + 1.5 * IQR))]
      if not outliers.empty:
        #df.loc[outliers.index, col] = winsorize(outliers[col], limits=[0.08, 0.08])
        df.drop(outliers.index, inplace=True)
```

Data preparation

LOF (Density-Based Anomaly Detection):

```
from sklearn.neighbors import LocalOutlierFactor

# Create the LOF model
model = LocalOutlierFactor(n_neighbors=5)

# Use the model to predict the outlier scores for each row
scores = model.fit_predict(df)

# Identify the outlier rows (those with a negative score) and remove them
outliers = df[scores == -1]
if not outliers.empty:
  df.drop(outliers.index, inplace=True)
```

Data preparation

Transformation:

Normalize the attributes using z-score

$$z = \frac{x - \mu}{\sigma}$$

μ: mean, σ: standard deviation

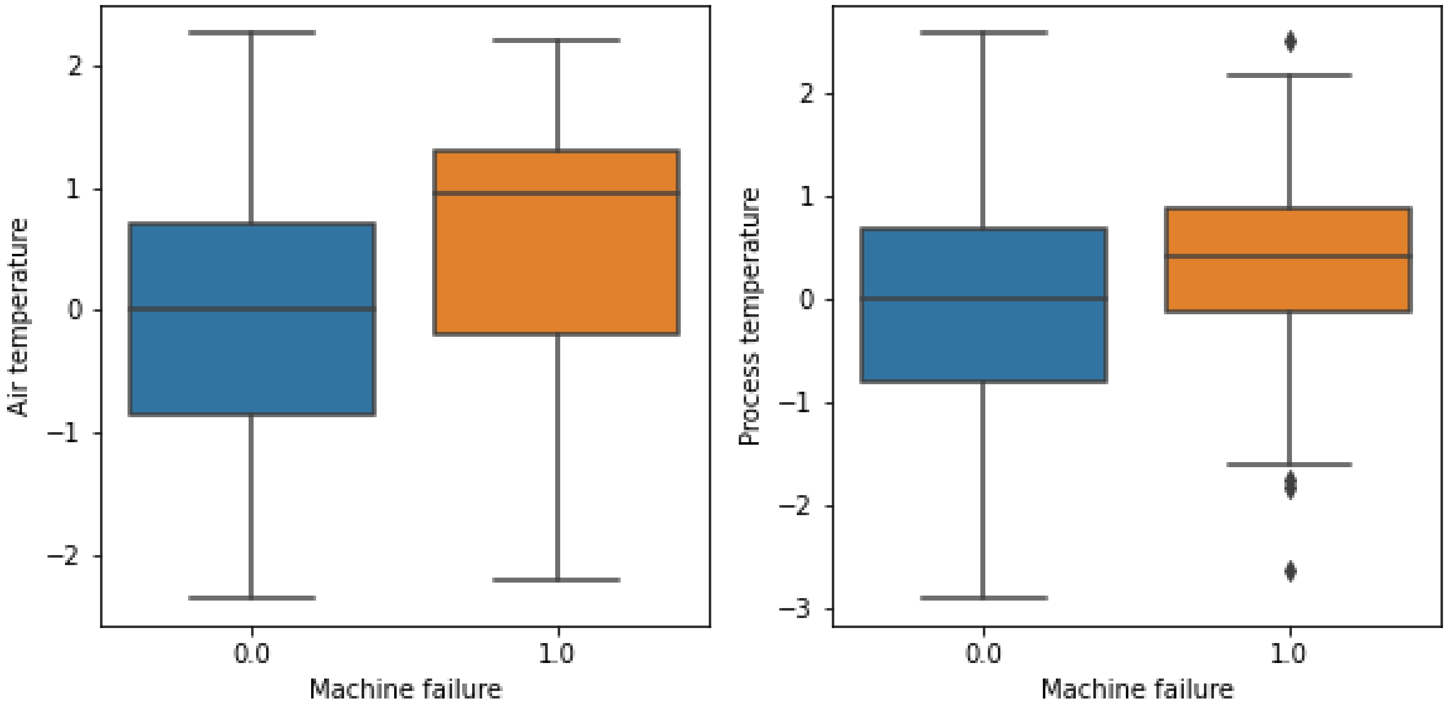
Data preparation

Transformation:

```
from scipy.stats import zscore

# Iterate over the columns in the dataframe
for col in df.columns:
  if col not in excluded_columns:
    # Normalize the values in the column
    df[col] = zscore(df[col])
```

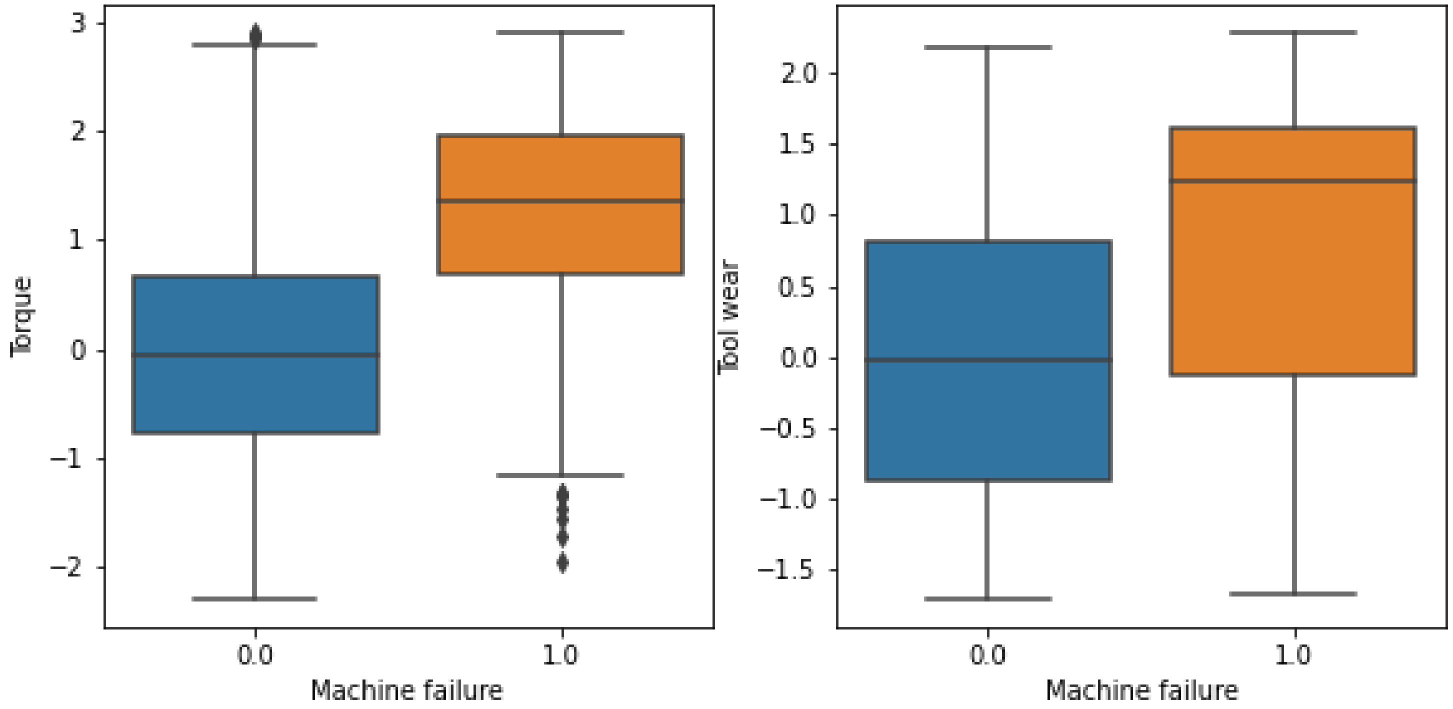
Data preparation - More visualizations

Box and Whisker plots for each attribute compared with Machine failure (target)



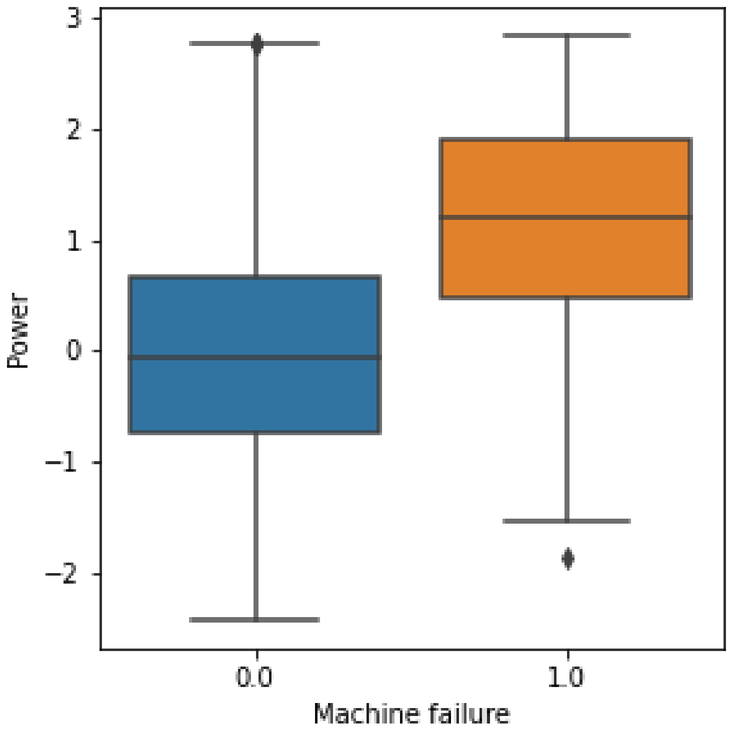
Data preparation - More visualizations

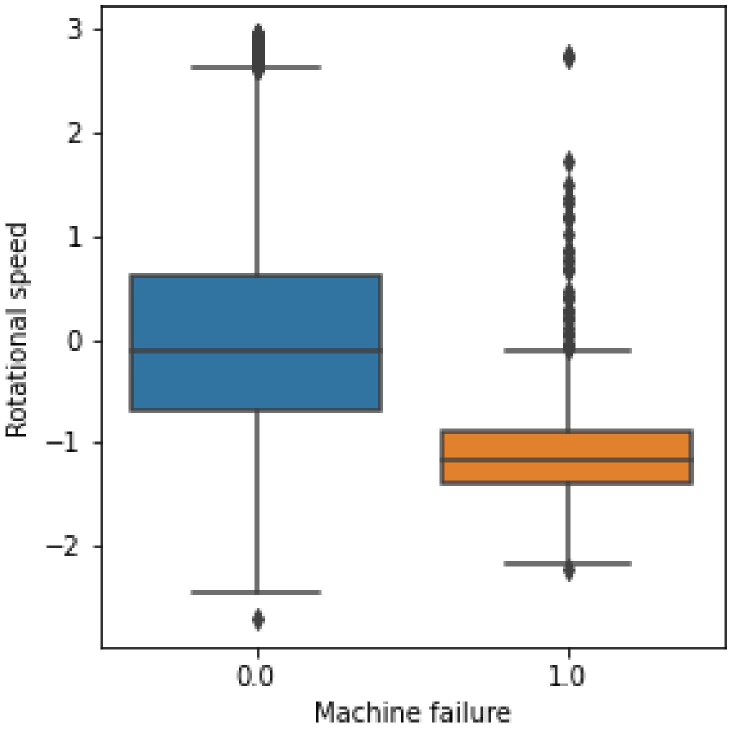
Box and Whisker plots for each attribute compared with Machine failure (target)



Data preparation - More visualizations

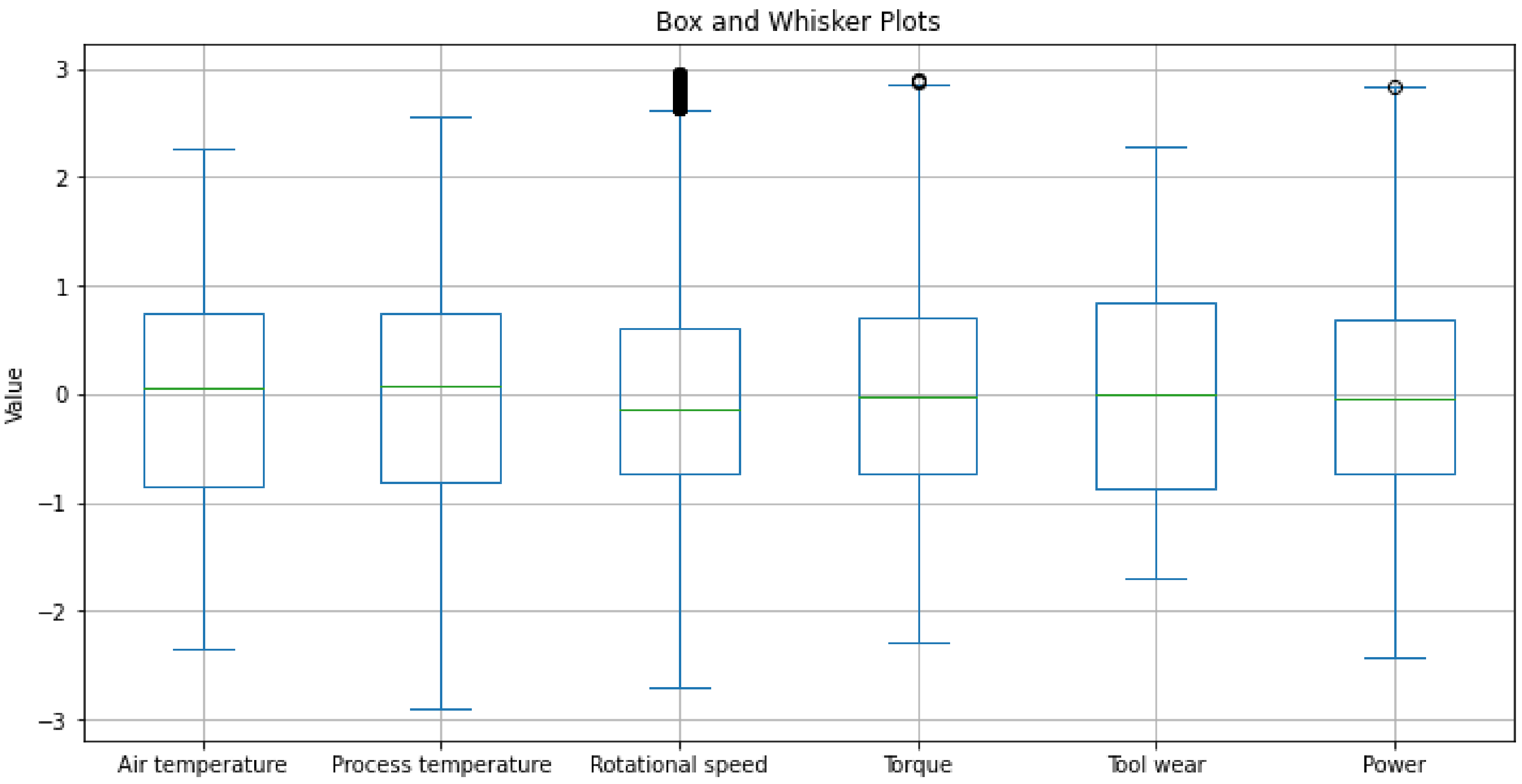
Box and Whisker plots for each attribute compared with Machine failure (target)





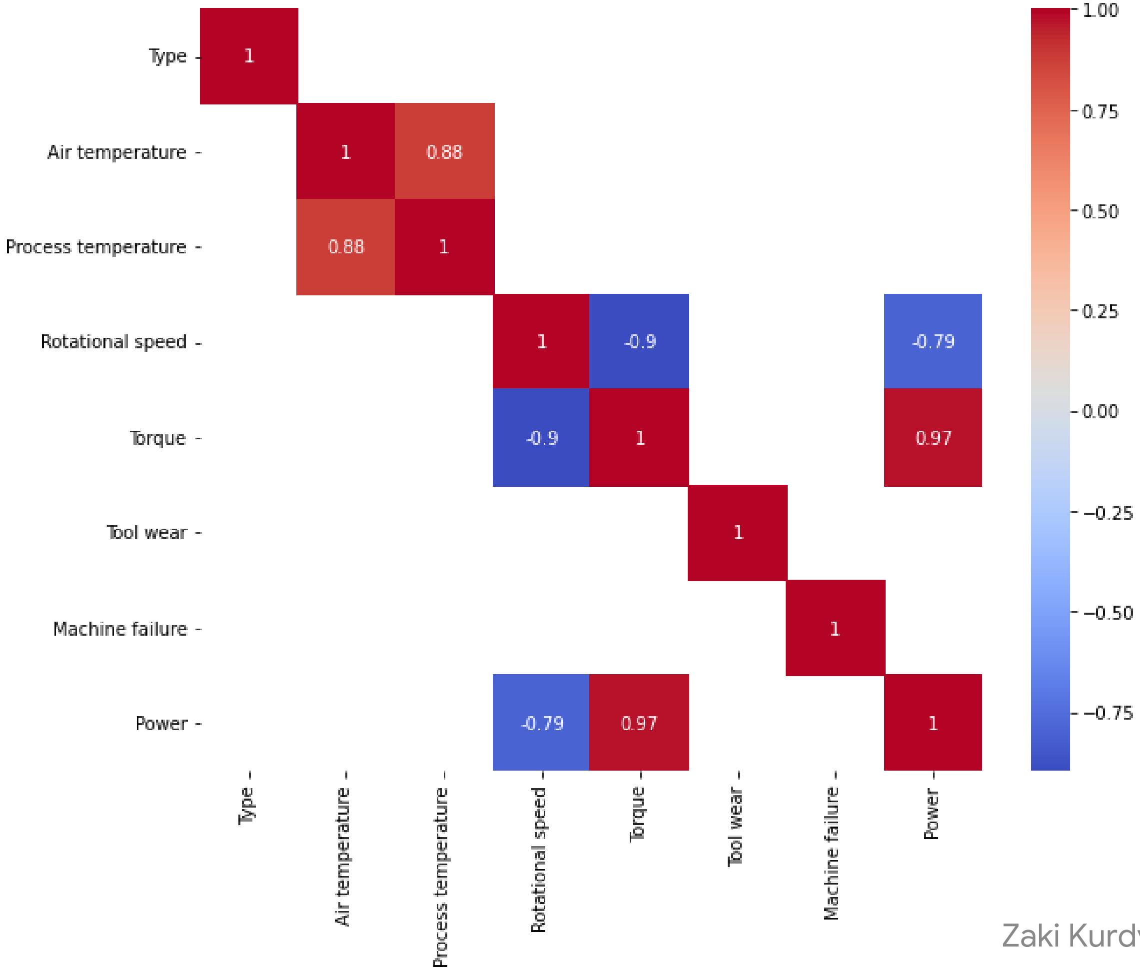
Data preparation - More visualizations

Box and Whisker plots for each attribute



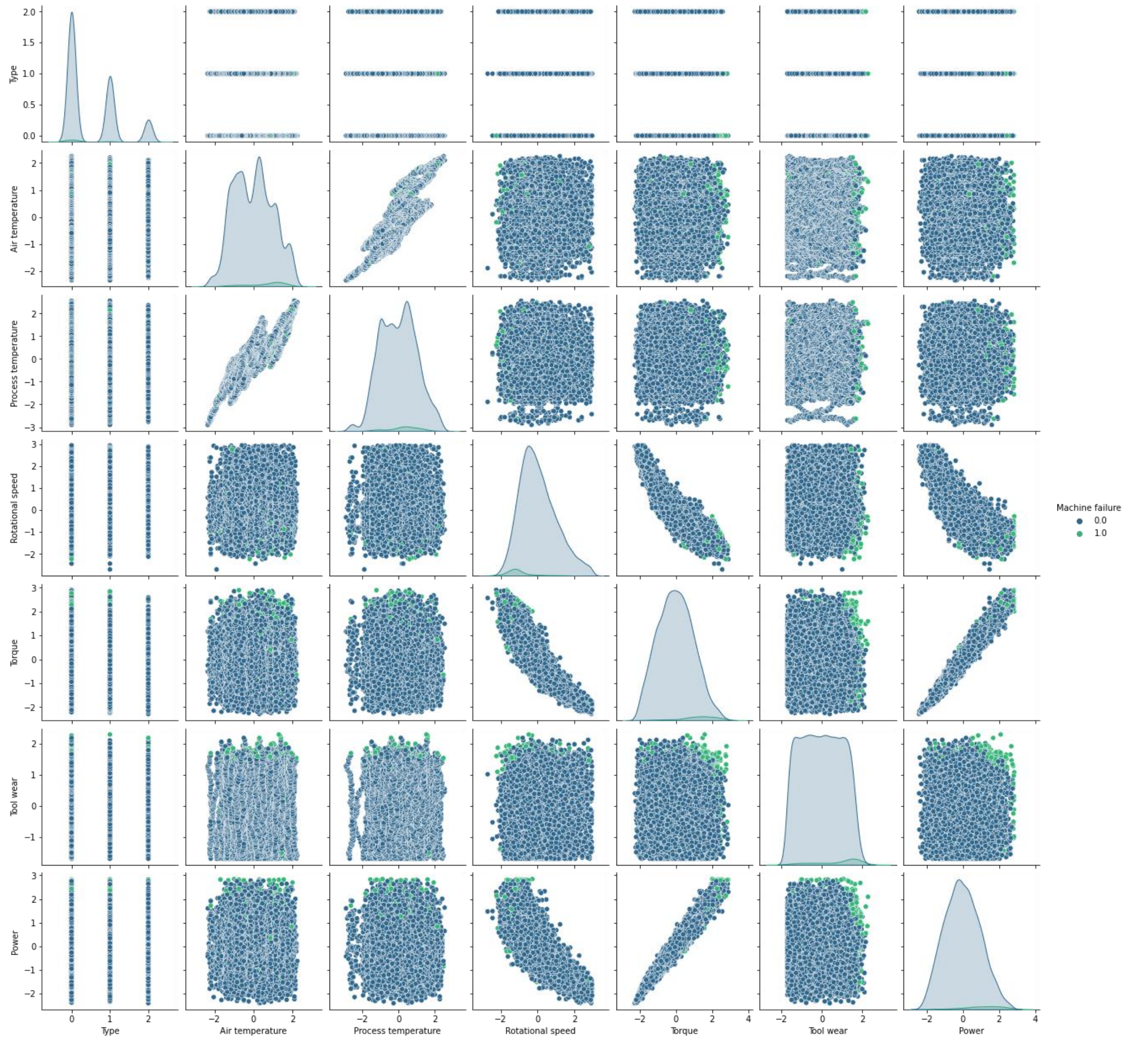
Data preparation - More visualizations

Correlation between the attributes with threshold = 0.3



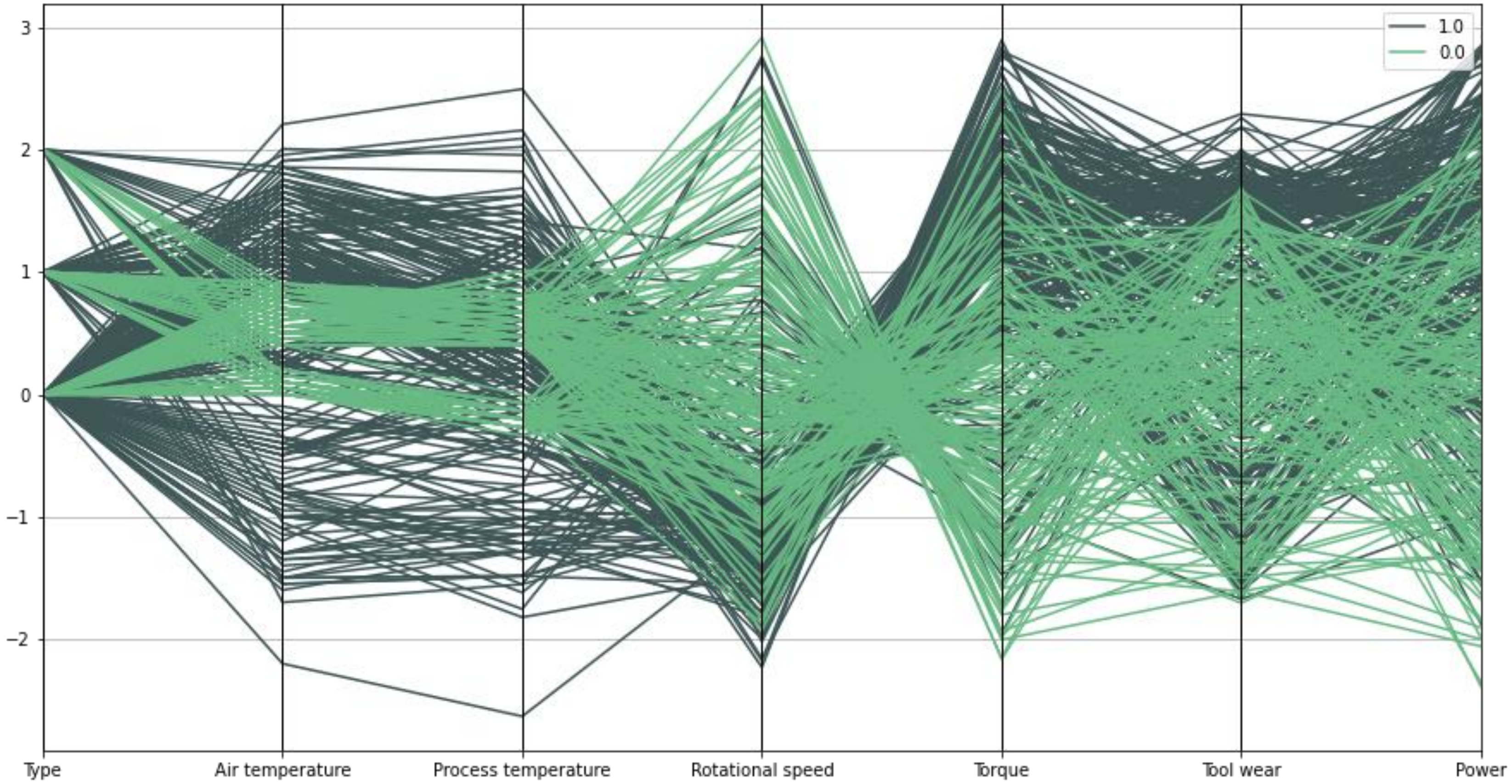
Data preparation - More visualizations

A scatter plot matrix (pair plot) to display the relationships between attributes in the dataset



Data preparation - More visualizations

Parallel coordinate plot (multi-dimensional view)



Data preparation

Data splitting:

Part	# of records	% percentage
Train	6850	70%
Test	2820	30%

Data preparation

Data splitting:

```python
from sklearn.model_selection import train_test_split

X = df.drop(["Machine failure"=], axis=1)
y = df["Machine failure"]

# Split the data into training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 0.3,
                                                    random_state = 0,
                                                    stratify = y)
```

Data preparation

Data sampling:

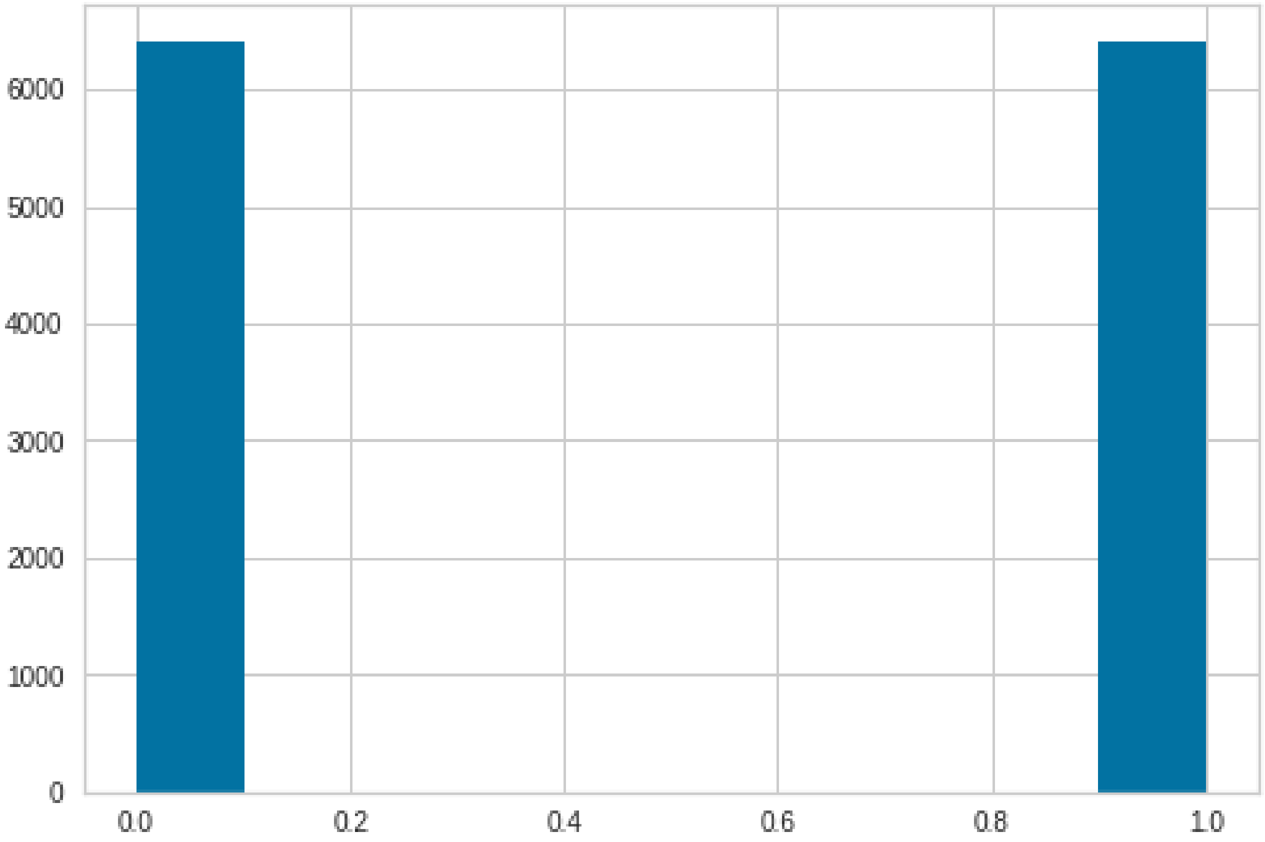
Because the data is imbalanced, we oversample the training set.

```
from imblearn.over_sampling import RandomOverSampler

oversample = RandomOverSampler(random_state=0)

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

Data preparation

Data sampling:

Training set after oversampling.

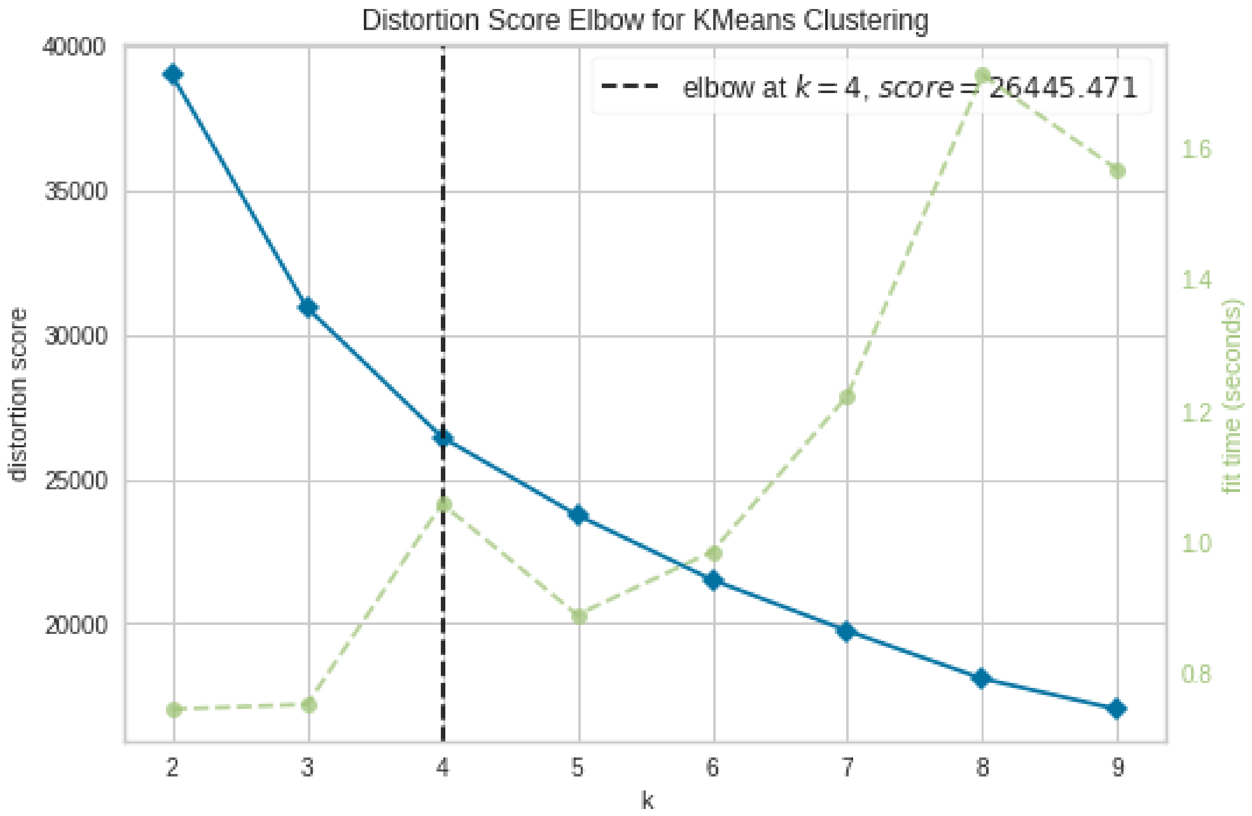


Descriptive Analytics

Descriptive analytics

Partitional Clustering, **K-means algorithm**

After using the elbow method, it turns out that the optimal number (k) of clusters is 4

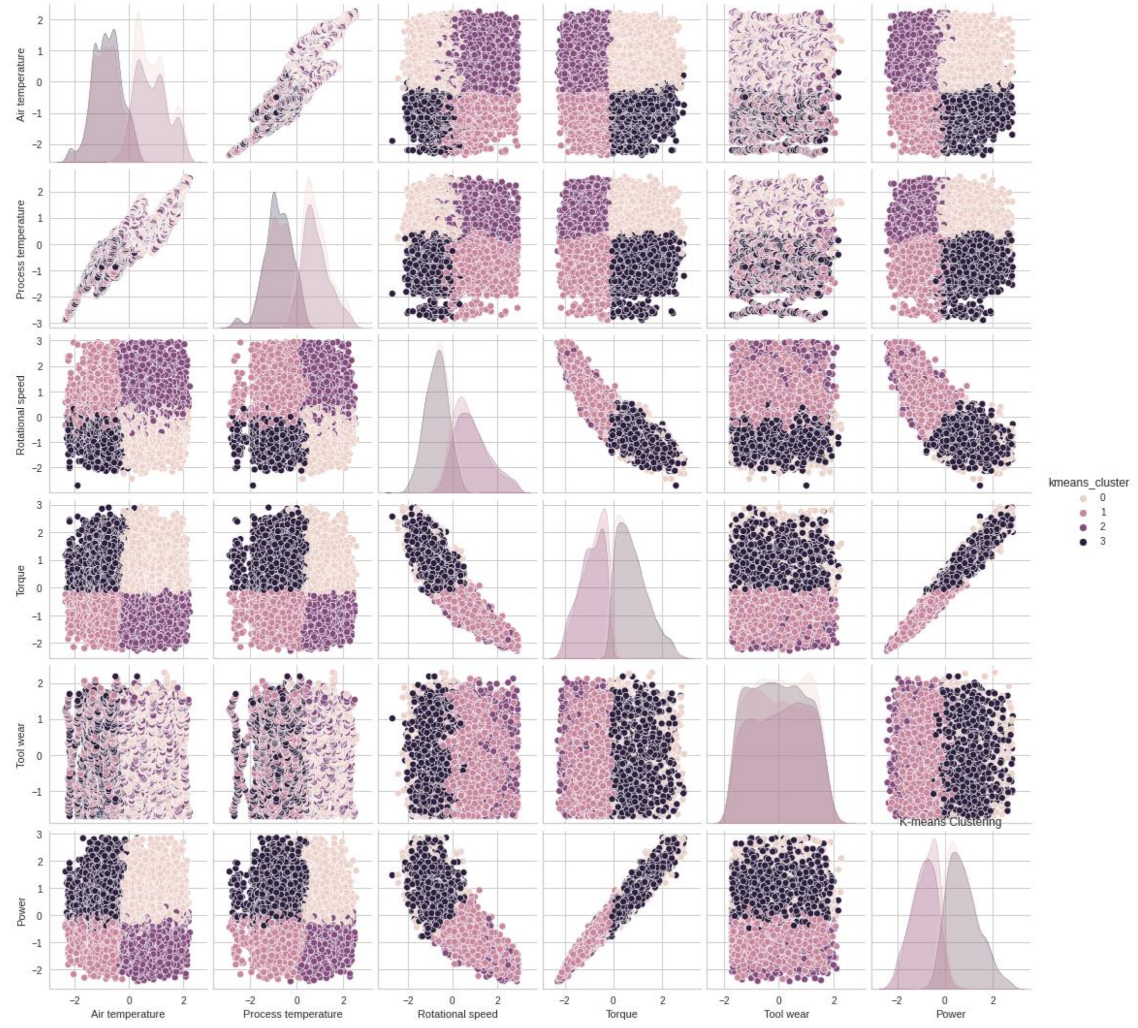


Descriptive analytics

Partitional Clustering, **K-means algorithm**

```python
from sklearn.cluster import KMeans

# K-means clustering
kmeans = KMeans(init="random",  n_clusters=4,
                n_init=10, max_iter=300, random_state=42)
kmeans.fit(X)

df["kmeans_cluster"] = kmeans.predict(X)
```

Descriptive analytics



Partitional Clustering,

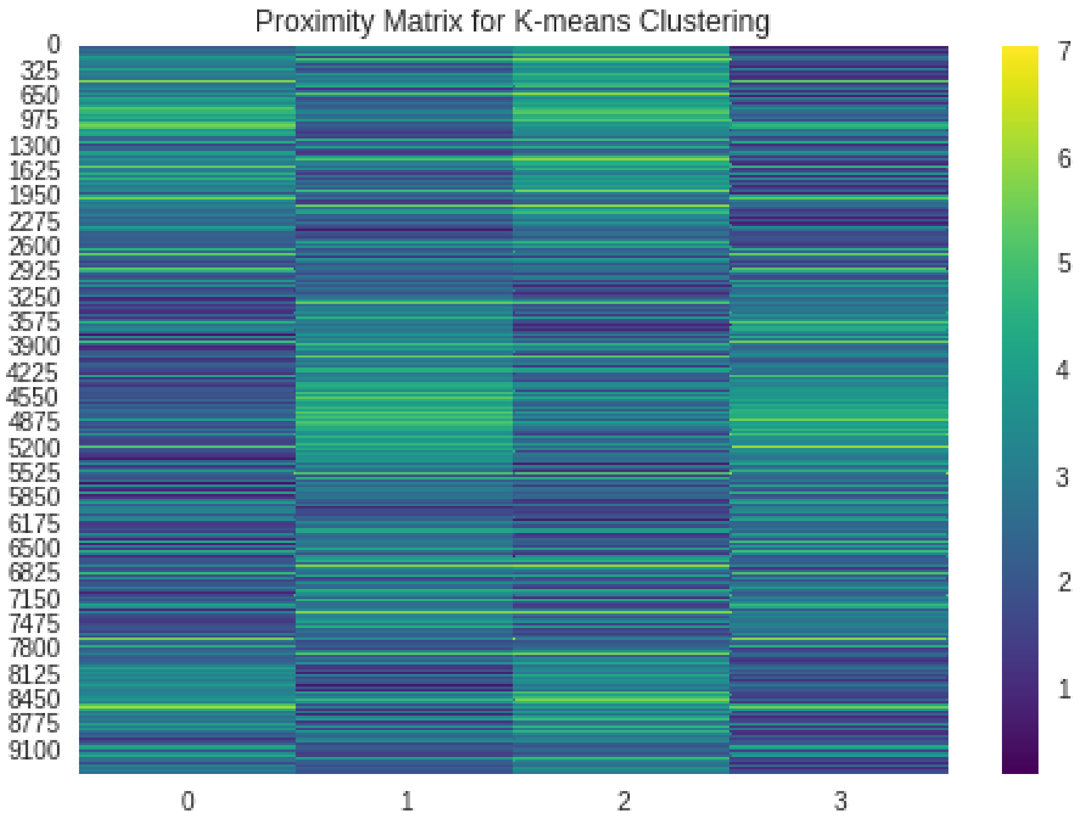
K-means algorithm

Pairplot of the data, colored by cluster label.

Descriptive analytics

Partitional Clustering, **K-means algorithm**

As shown in the proximity matrices, the clusters are not so crisp.

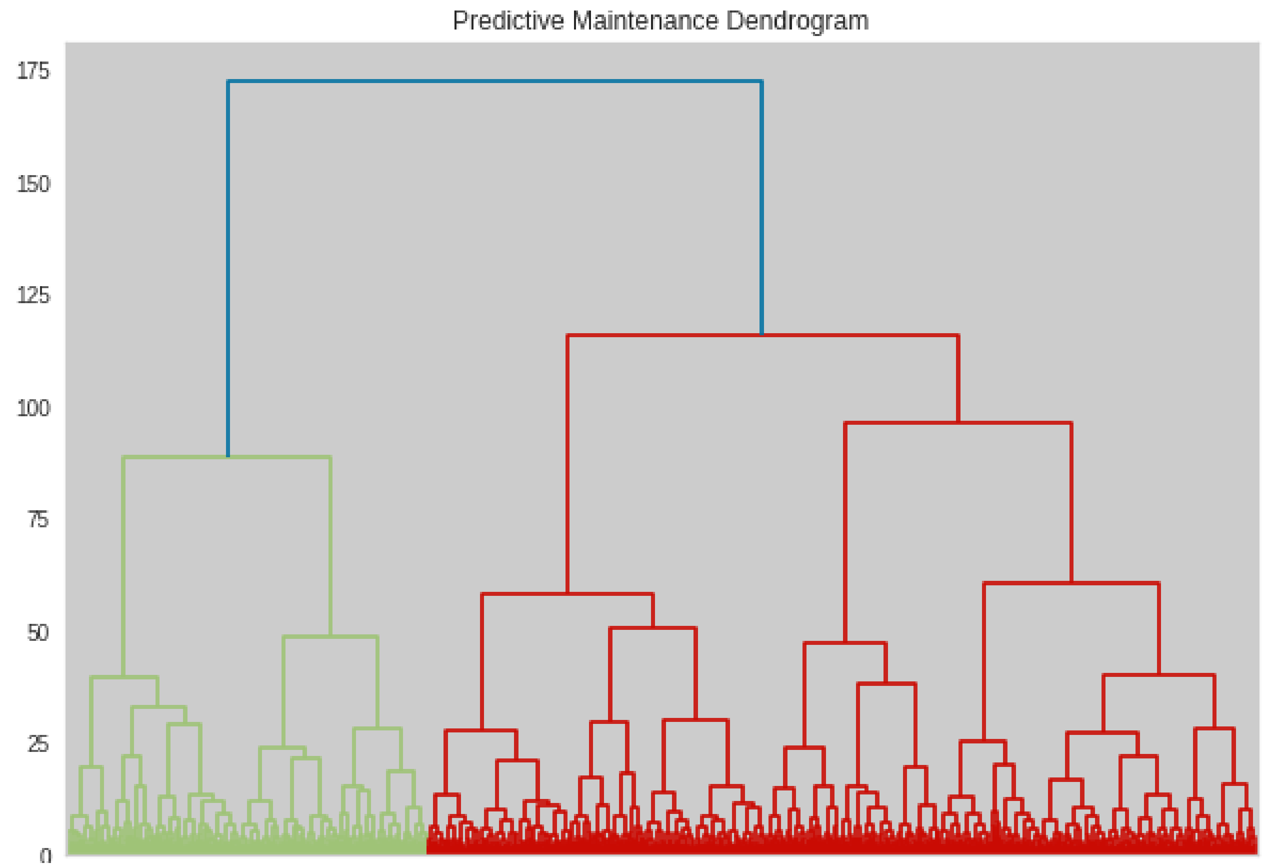


Silhouette Coefficient: 0.225

Descriptive analytics

Hierarchical clustering, **Agglomerative**

Dendrogram

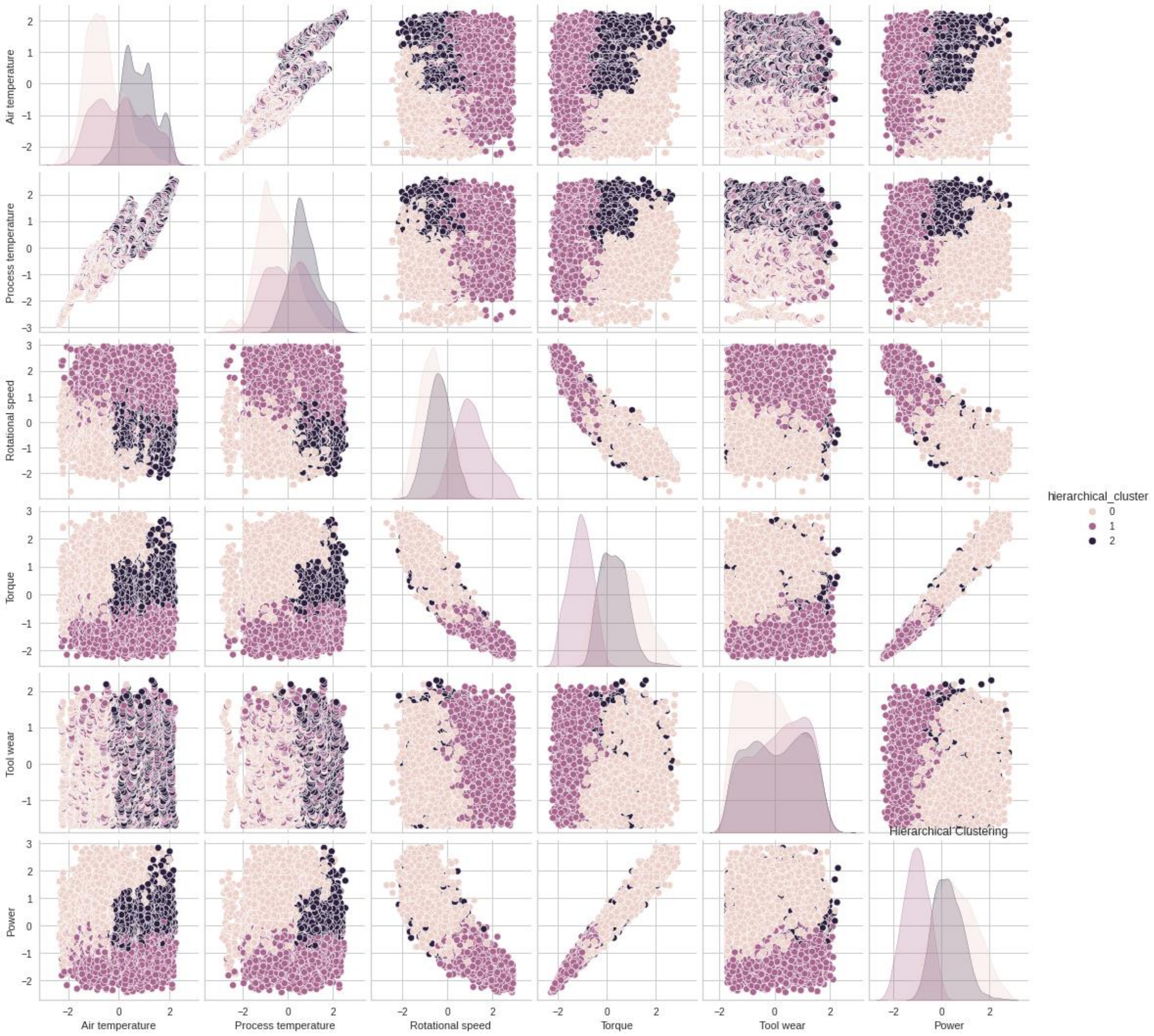


Descriptive analytics

Hierarchical clustering, **Agglomerative**

```
from sklearn.cluster import AgglomerativeClustering

# Hierarchical clustering
model = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='ward')
model.fit(X)
df["hierarchical_cluster"] = model.labels_
```

Descriptive analytics



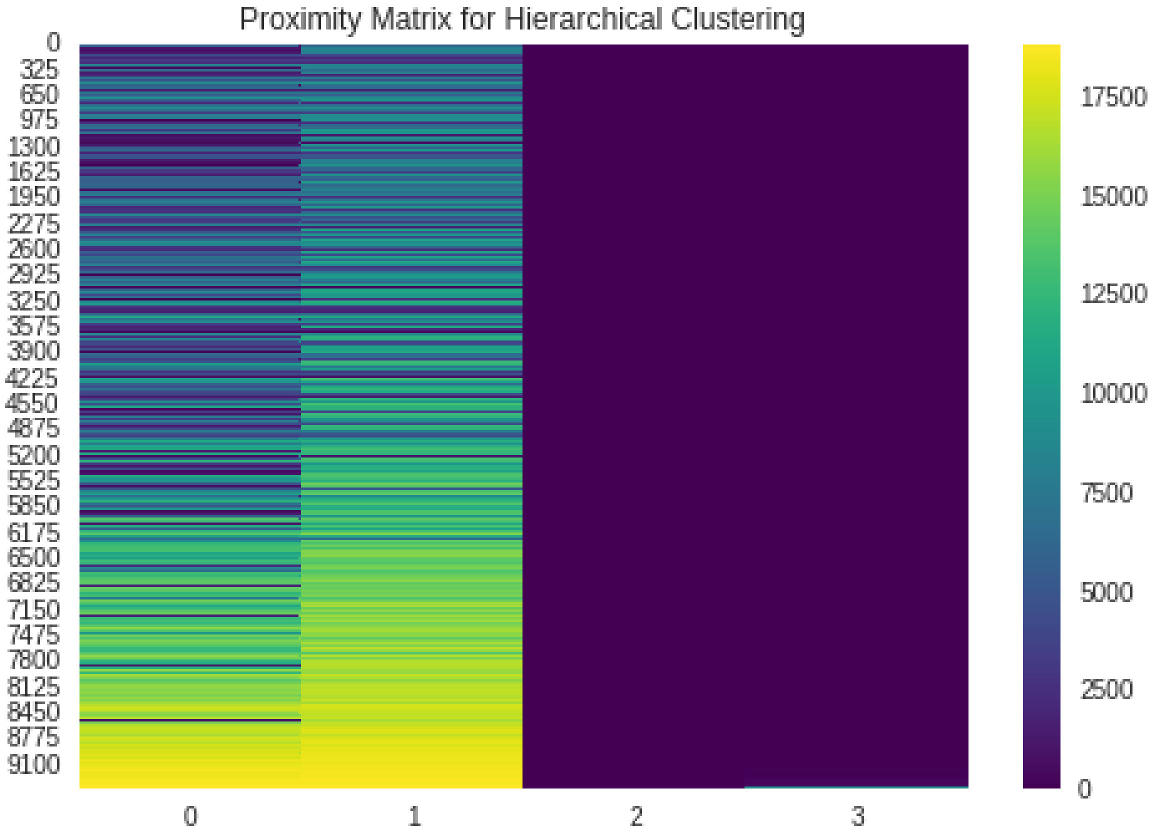
Hierarchical clustering, **Agglomerative**

Pairplot of the data, colored by cluster label.

Descriptive analytics

Hierarchical clustering, **Agglomerative**

Silhouette Coefficient: 0.180

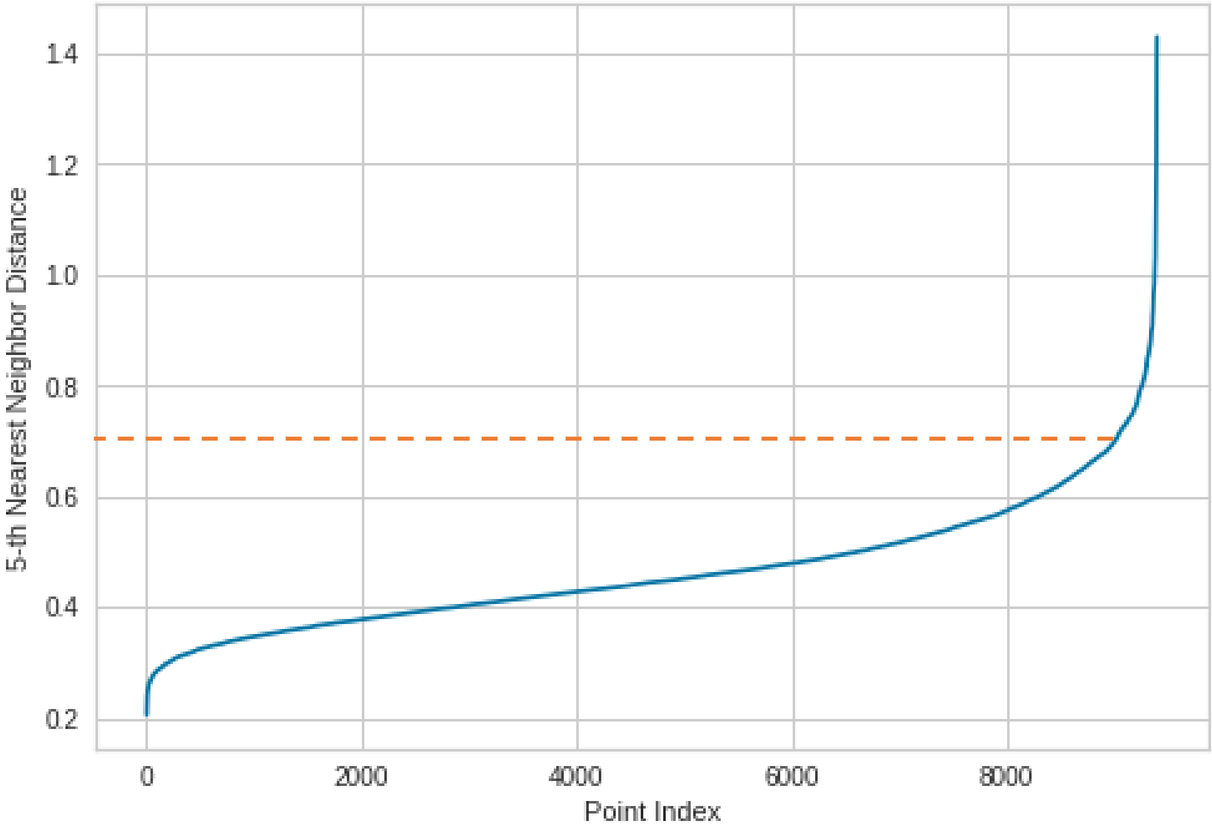


Descriptive analytics

Density-based clustering, **DBSACN**

I use the **nearest neighbors** model to find the average distance between data points to determine the best value for **EPS** when **MinPts** = 5

EPS = 0.7



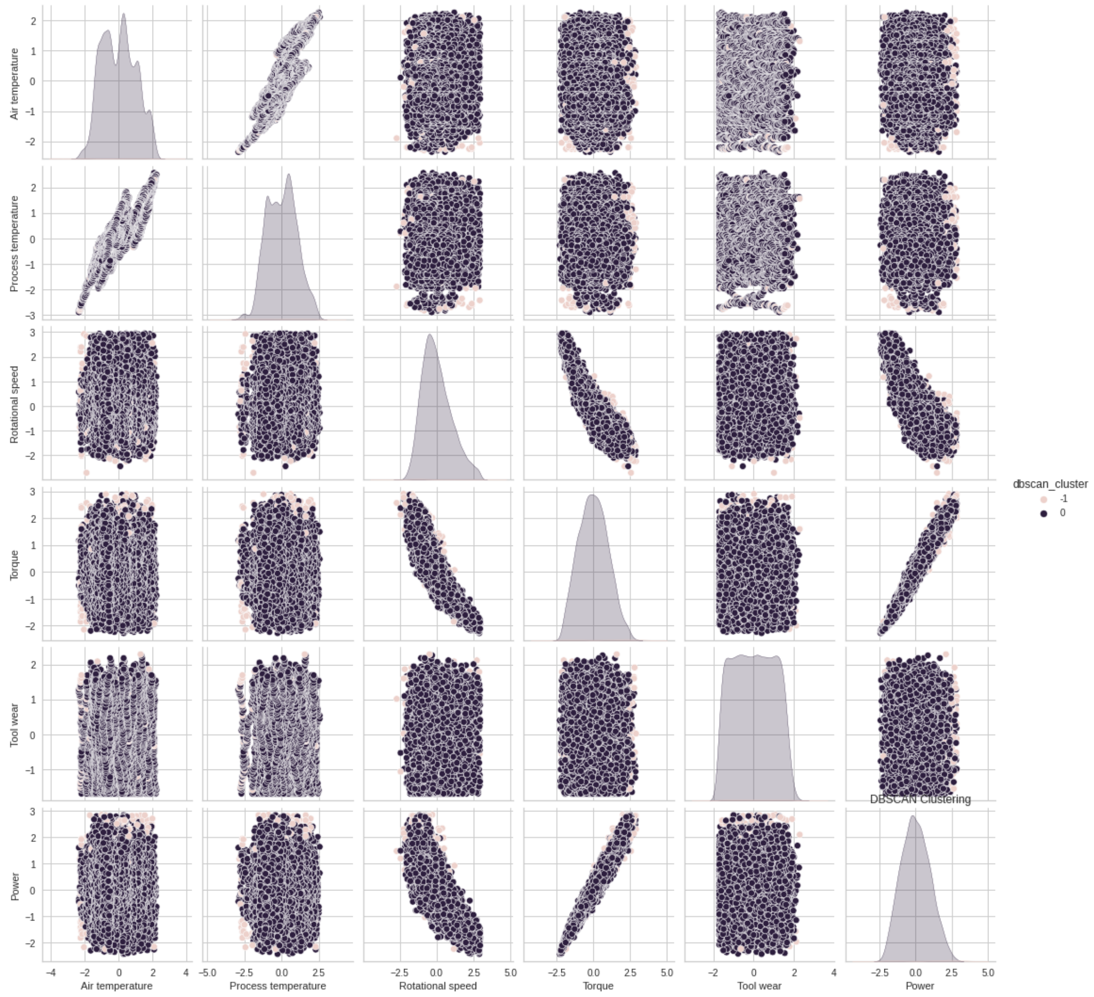
Descriptive analytics

Density-based clustering, **DBSACN**

Silhouette Coefficient: 0.292

```python
from sklearn.cluster import DBSCAN

# create a DBSCAN model
model = DBSCAN(eps=0.7, min_samples=5)
model.fit(X)

# obtain the cluster labels
df['dbscan_cluster'] = model.labels_
```

Descriptive analytics



Density-based clustering, **DBSACN**

Pairplot of the data, colored by cluster label.

Predictive Analytics

Predictive analytics

First I define a function to calculate and store model scores

```
from sklearn.metrics import f1_score, precision_score, recall_score,
                            accuracy_score, classification_report
import time

model_performance = pd.DataFrame(columns=['Accuracy', 'Precision',
                                          'Recall', 'F1-Score', 'Training time',
                                          'Prediction time'])


def log_scores(model_name, y_test, y_predictions):
  accuracy = accuracy_score(y_test, y_predictions)
  precision = precision_score(y_test, y_predictions, average='weighted')
  recall = recall_score(y_test, y_predictions, average='weighted')
  precision = precision_score(y_test, y_predictions, average='weighted')
  f1 = f1_score(y_test, y_predictions, average='weighted')

  # save the scores in model_performance dataframe
  model_performance.loc[model_name] = [accuracy, precision, recall, f1,
                                       end_train-start, end_predict-end_train]
```

Predictive analytics

Decision Tree Model - Build

```
from sklearn.tree import DecisionTreeClassifier

start = time.time()
model = DecisionTreeClassifier(max_depth = 8).fit(X_train, y_train)
end_train = time.time()
y_predictions = model.predict(X_test)
end_predict = time.time()

# evaluate the model
log_scores("Decision Tree", y_test, y_predictions)
```

Predictive analytics

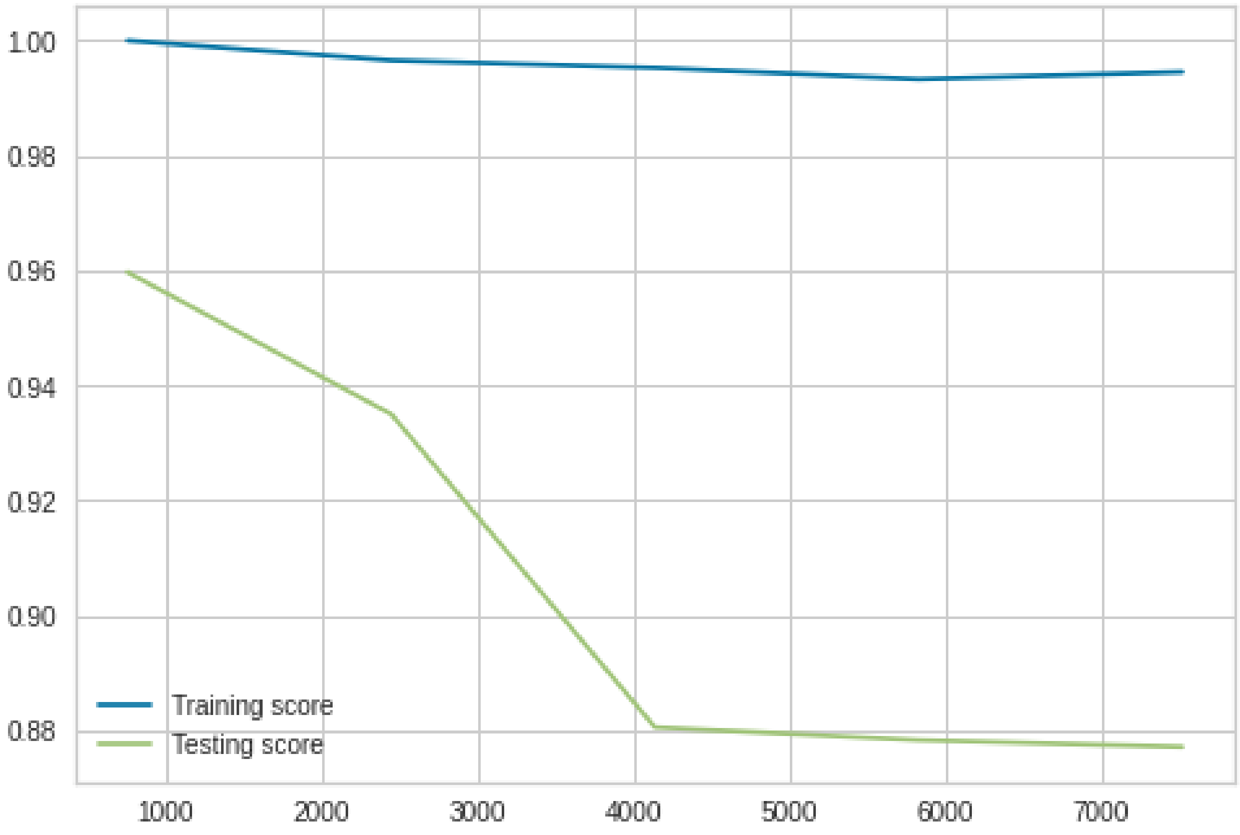
Decision Tree Model – classification report

Decision Tree

	precision	recall	f1-score	support
0.0	0.99	0.95	0.97	2746
1.0	0.24	0.61	0.35	74
accuracy			0.94	2820
macro avg	0.62	0.78	0.66	2820
weighted avg	0.97	0.94	0.95	2820

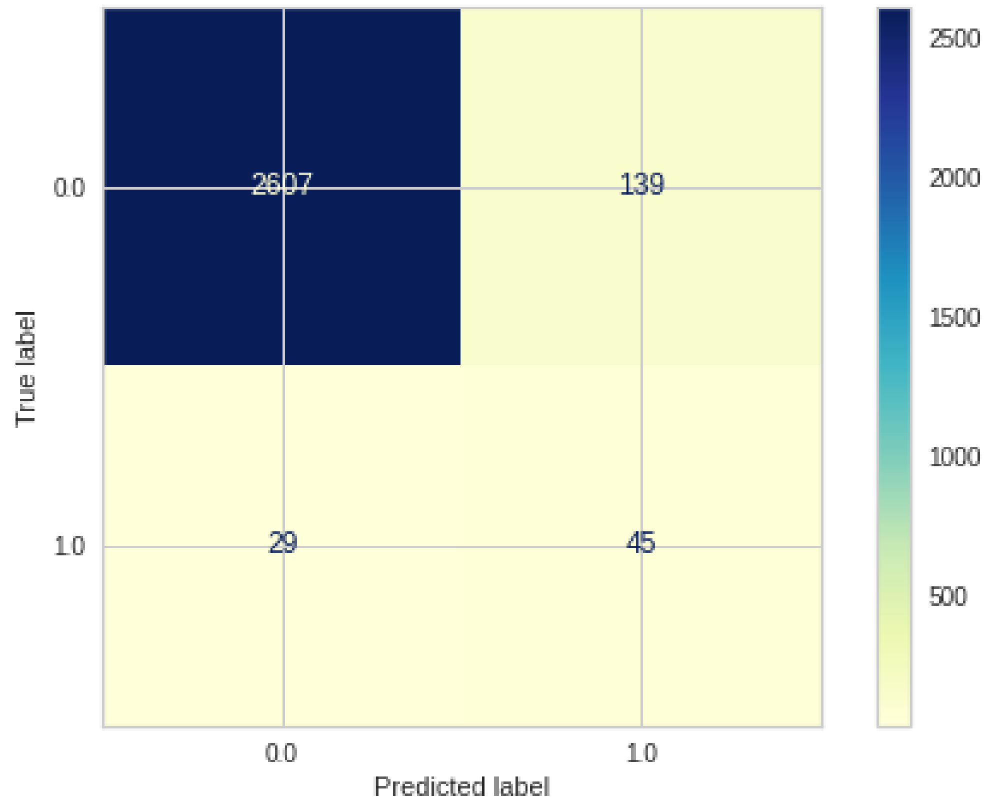
Predictive analytics

Decision Tree Model – learning curve



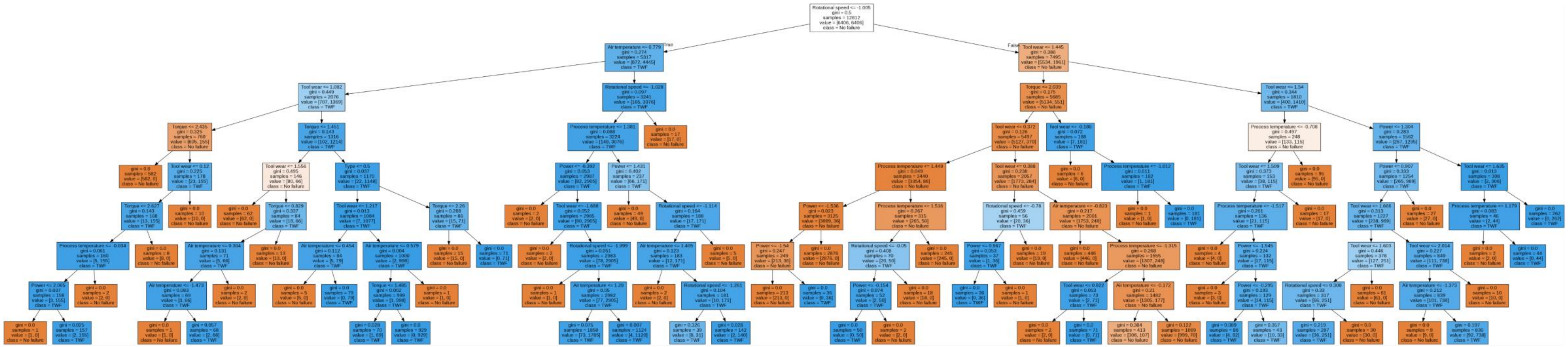
Predictive analytics

Decision Tree Model – confusion matrix



Predictive analytics

Decision Tree Model – nodes



Predictive analytics

k-NN (K-nearest neighbors) Model

Grid search to find the best value for n_neighbors

'n_neighbors' : 2

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV

# create the model
knn = KNeighborsClassifier()

# define the parameter grid
param_grid = {'n_neighbors': range(2, 20)}

# create the grid search object
grid_search = GridSearchCV(knn, param_grid,
                           cv=5, scoring='accuracy')

# fit the grid search to the data
grid_search.fit(X_train, y_train)

# print the best parameters
print(grid_search.best_params_)
```

Predictive analytics

k-NN (K-nearest neighbors) Model - Build

```
start = time.time()
model = KNeighborsClassifier(n_neighbors=2).fit(X_train, y_train)
end_train = time.time()
y_predictions = model.predict(X_test) # predictions from the testset
end_predict = time.time()

# evaluate the model
log_scores("k-NN", y_test, y_predictions)
```

Predictive analytics

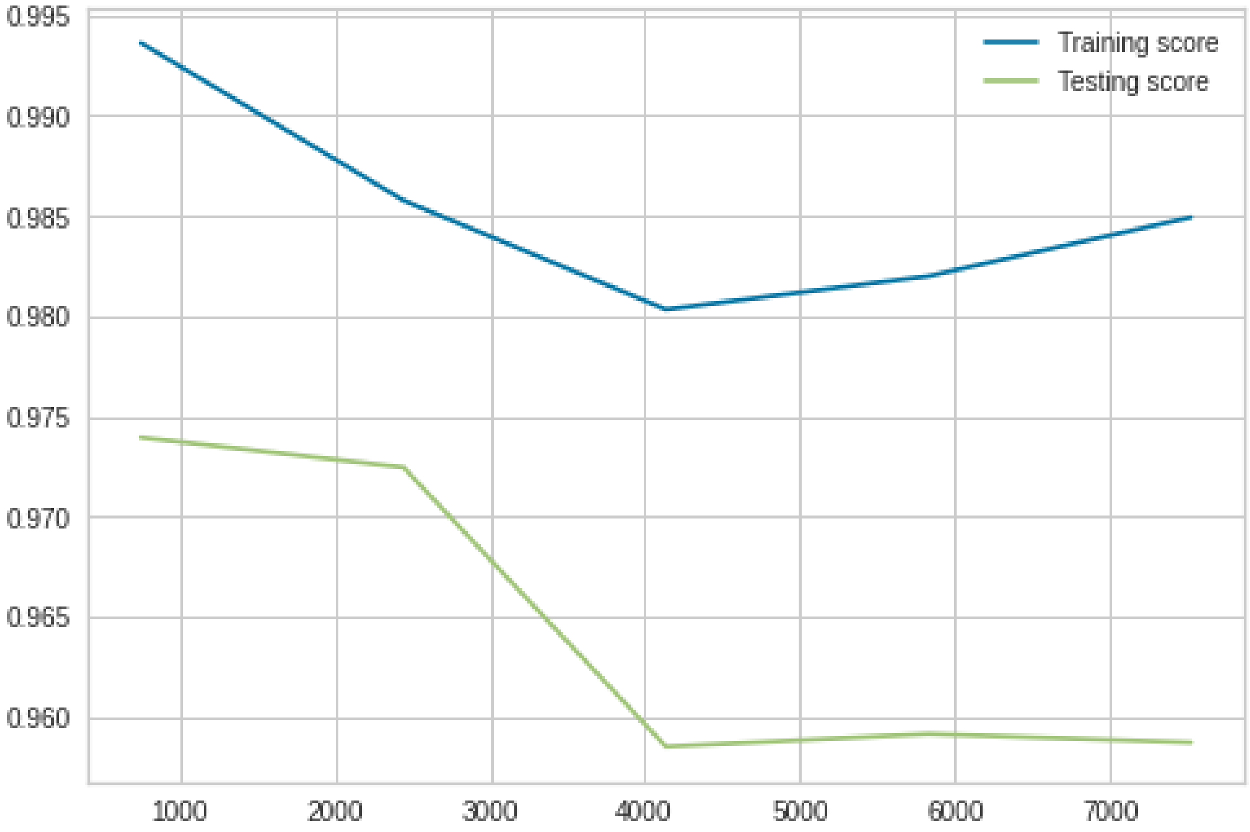
k-NN Model – classification report

k-NN Model

	precision	recall	f1-score	support
0.0	0.98	0.99	0.99	2746
1.0	0.50	0.36	0.42	74
accuracy			0.97	2820
macro avg	0.74	0.68	0.70	2820
weighted avg	0.97	0.97	0.97	2820

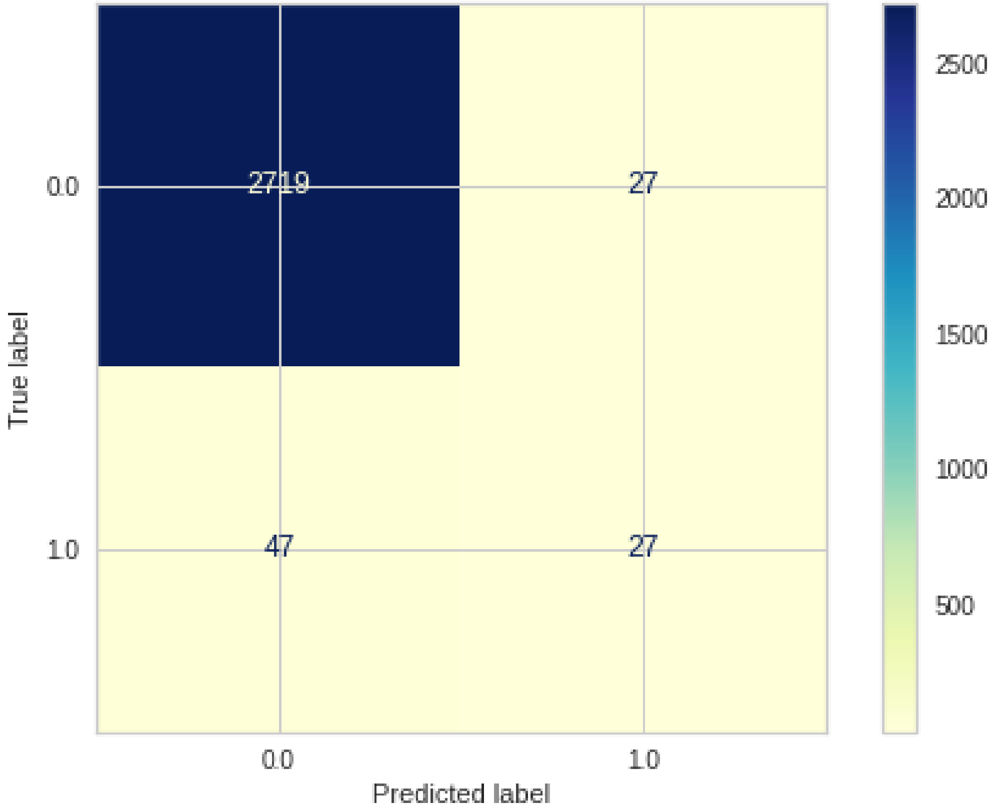
Predictive analytics

k-NN Model – learning curve



Predictive analytics

k-NN Model – confusion matrix



Predictive analytics

Random Forest Model - Build

```
from sklearn.ensemble import RandomForestClassifier

start = time.time()
model = RandomForestClassifier(n_estimators=100, n_jobs=-1,
                               random_state=0, bootstrap=True)
                              .fit(X_train, y_train)
end_train = time.time()
y_predictions = model.predict(X_test)
end_predict = time.time()

# evaluate the model
log_scores("Random Forest", y_test, y_predictions)
```

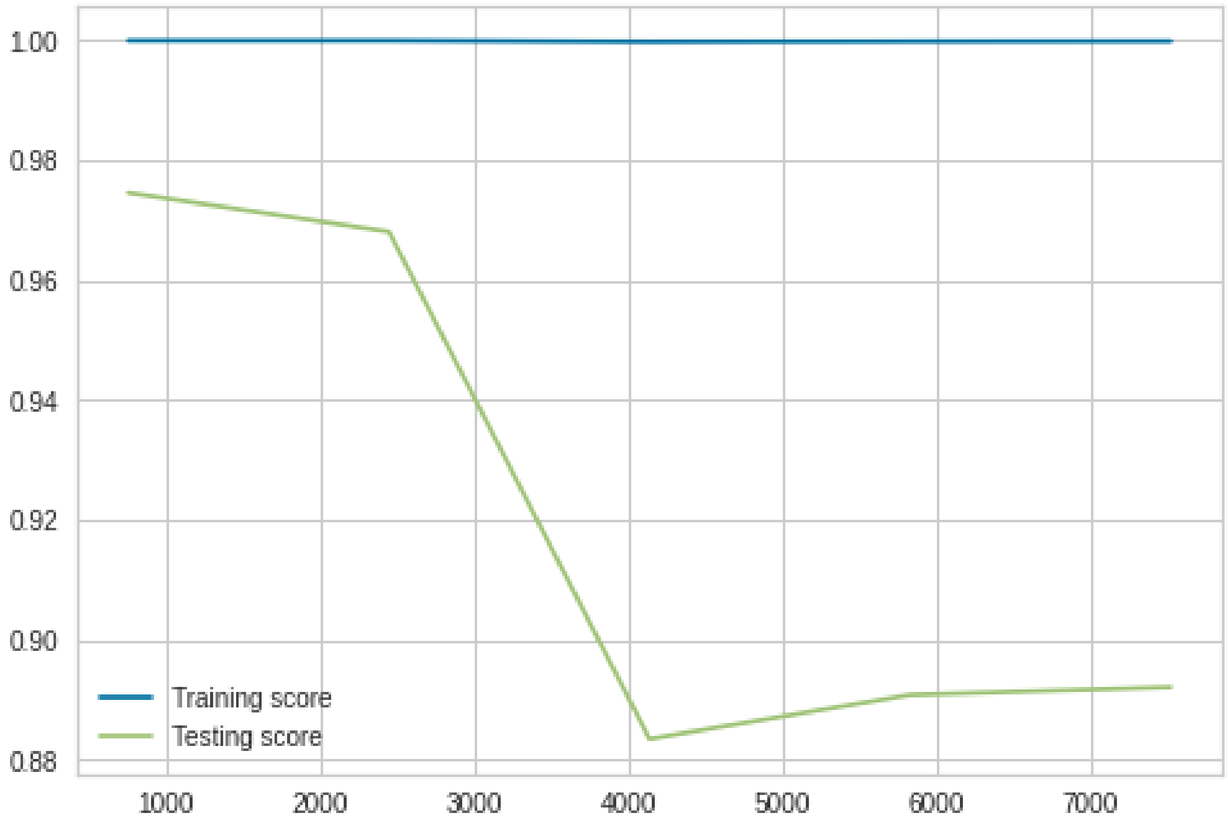
Predictive analytics

Random Forest Model – classification report

```
Random Forest Model
              precision    recall  f1-score   support

         0.0       0.99      1.00      0.99      2746
         1.0       0.76      0.53      0.62        74

    accuracy                           0.98      2820
   macro avg       0.88      0.76      0.81      2820
weighted avg       0.98      0.98      0.98      2820
```

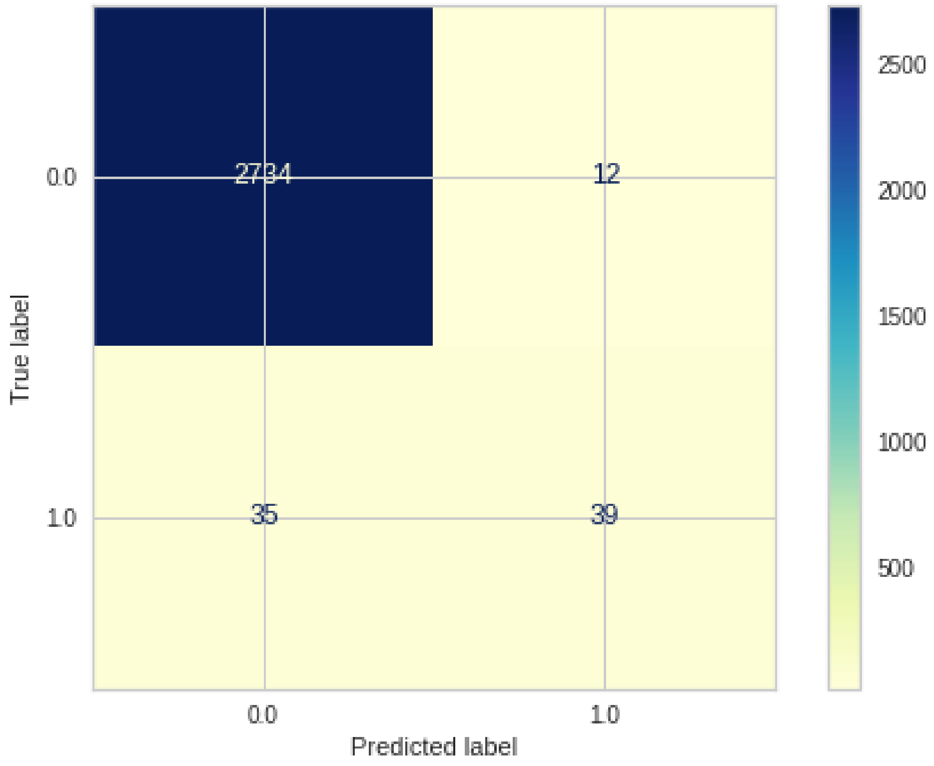
Predictive analytics

Random Forest Model – learning curve



Predictive analytics

Random Forest Model – confusion matrix



Predictive analytics

Gradient Boosting Model - Build

```
from sklearn.ensemble import GradientBoostingClassifier

start = time.time()
model = GradientBoostingClassifier().fit(X_train, y_train)
end_train = time.time()
y_predictions = model.predict(X_test)
end_predict = time.time()

# evaluate the model
log_scores("Gradient Boosting", y_test, y_predictions)
```

Predictive analytics

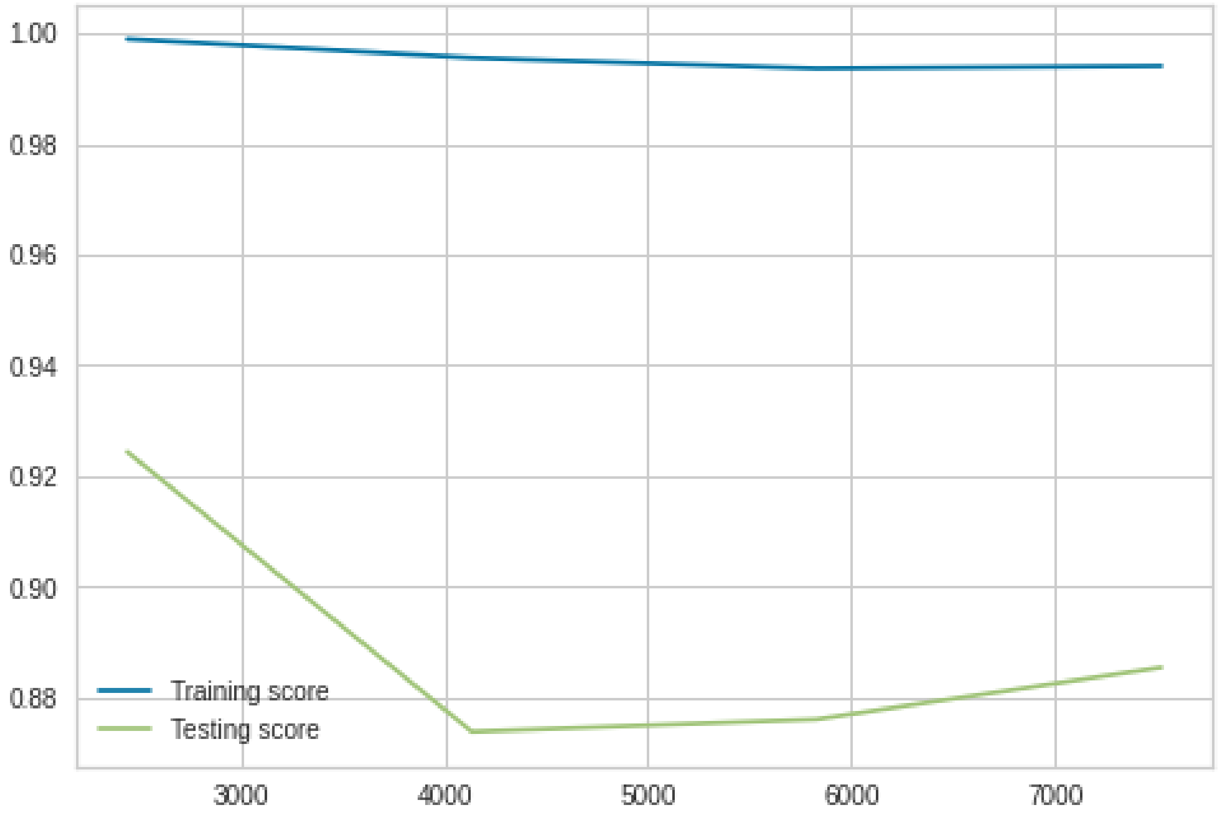
Gradient Boosting Model – classification report

Gradient Boosting

	precision	recall	f1-score	support
0.0	0.99	0.94	0.97	2746
1.0	0.27	0.81	0.41	74
accuracy			0.94	2820
macro avg	0.63	0.88	0.69	2820
weighted avg	0.98	0.94	0.95	2820

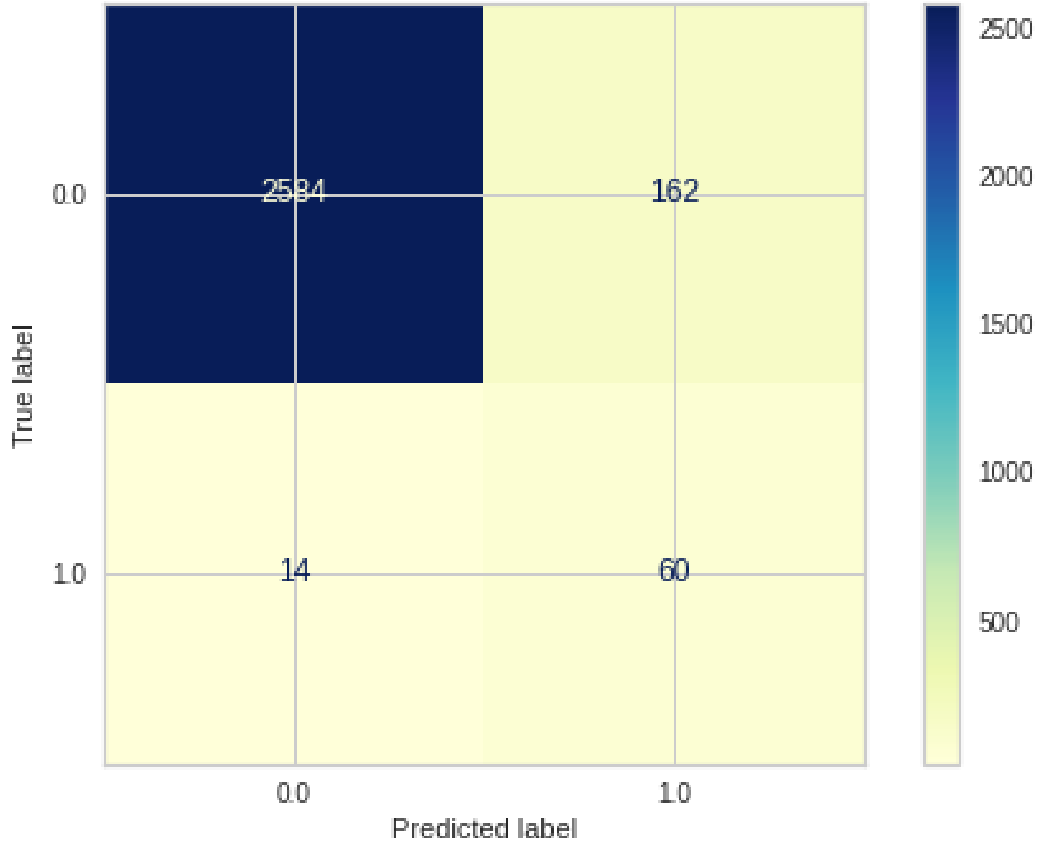
Predictive analytics

Gradient Boosting Model – learning curve



Predictive analytics

Gradient Boosting Model – confusion matrix



Predictive analytics

Gaussian Naive Bayes Model - Build

```
from sklearn.naive_bayes import GaussianNB

start = time.time()
model = GaussianNB().fit(X_train, y_train)
end_train = time.time()
y_predictions = model.predict(X_test)
end_predict = time.time()

# evaluate the model
log_scores("Gaussian Naive Bayes", y_test, y_predictions)
```

Predictive analytics

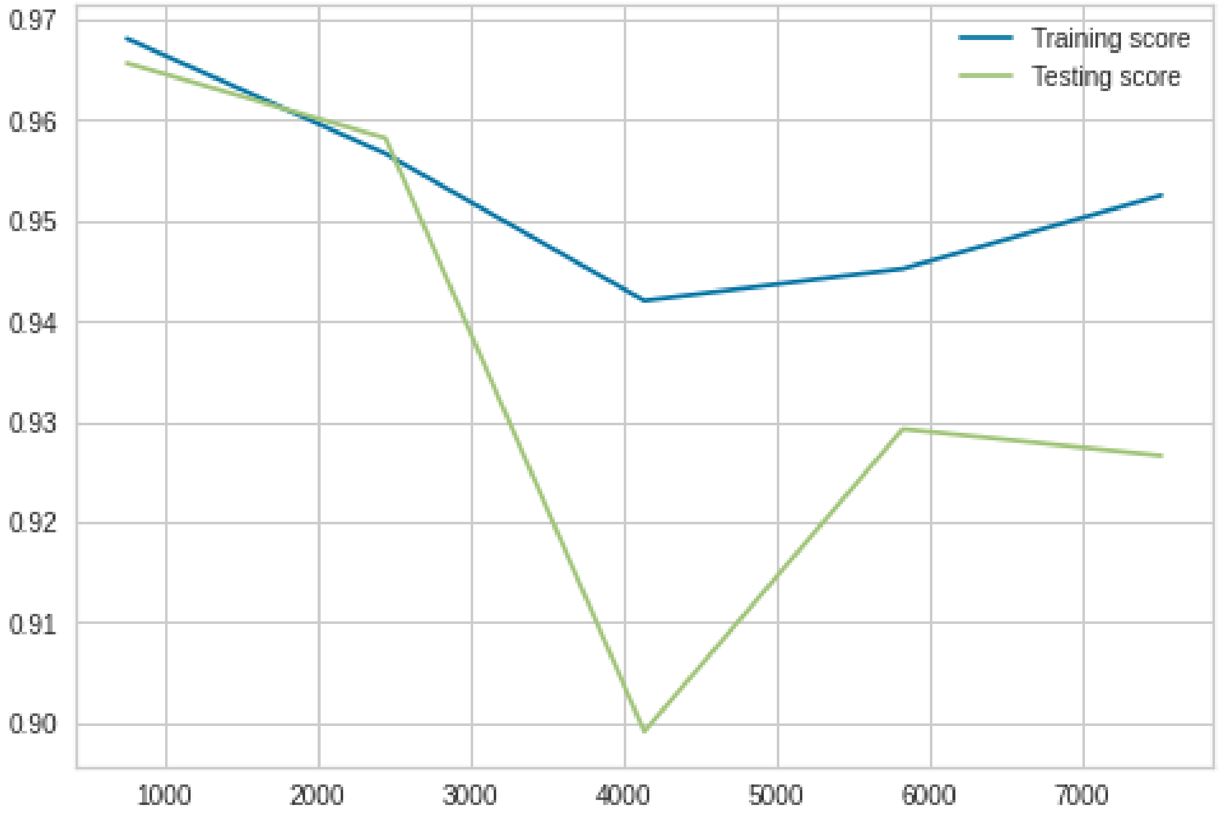
Gaussian Naive Bayes Model – classification report

Gaussian Naive Bayes

	precision	recall	f1-score	support
0.0	1.00	0.75	0.86	2746
1.0	0.09	0.92	0.17	74
accuracy			0.76	2820
macro avg	0.54	0.84	0.51	2820
weighted avg	0.97	0.76	0.84	2820

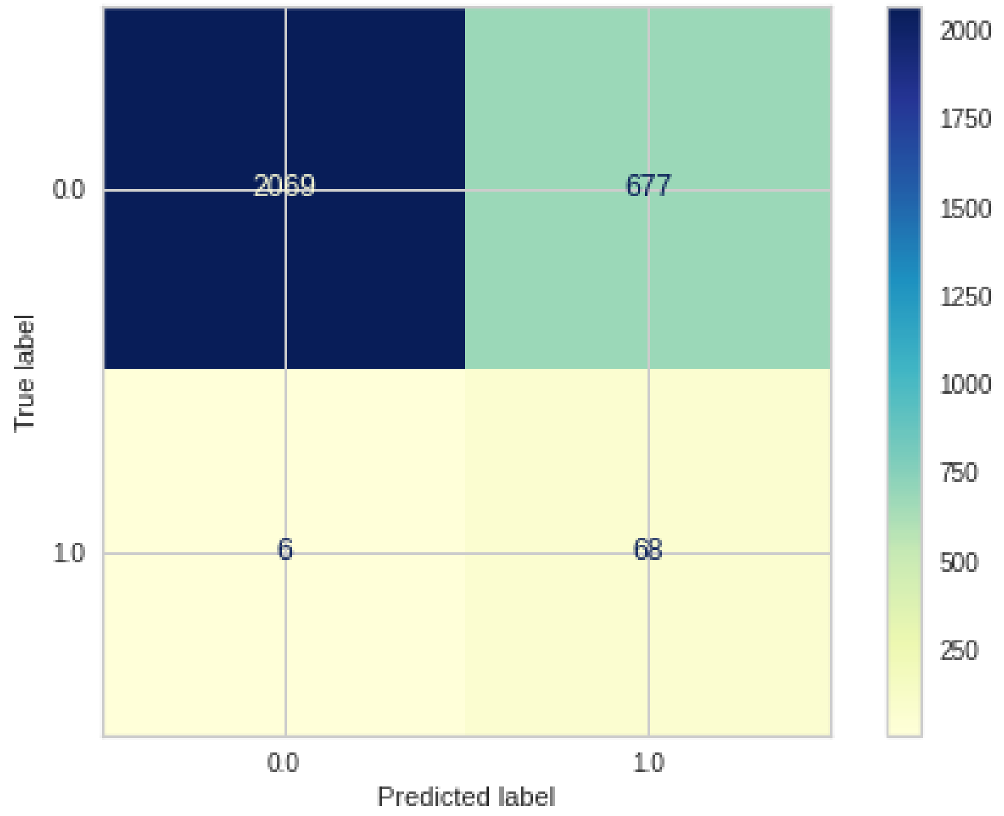
Predictive analytics

Gaussian Naive Bayes Model – learning curve



Predictive analytics

Gaussian Naive Bayes Model – confusion matrix



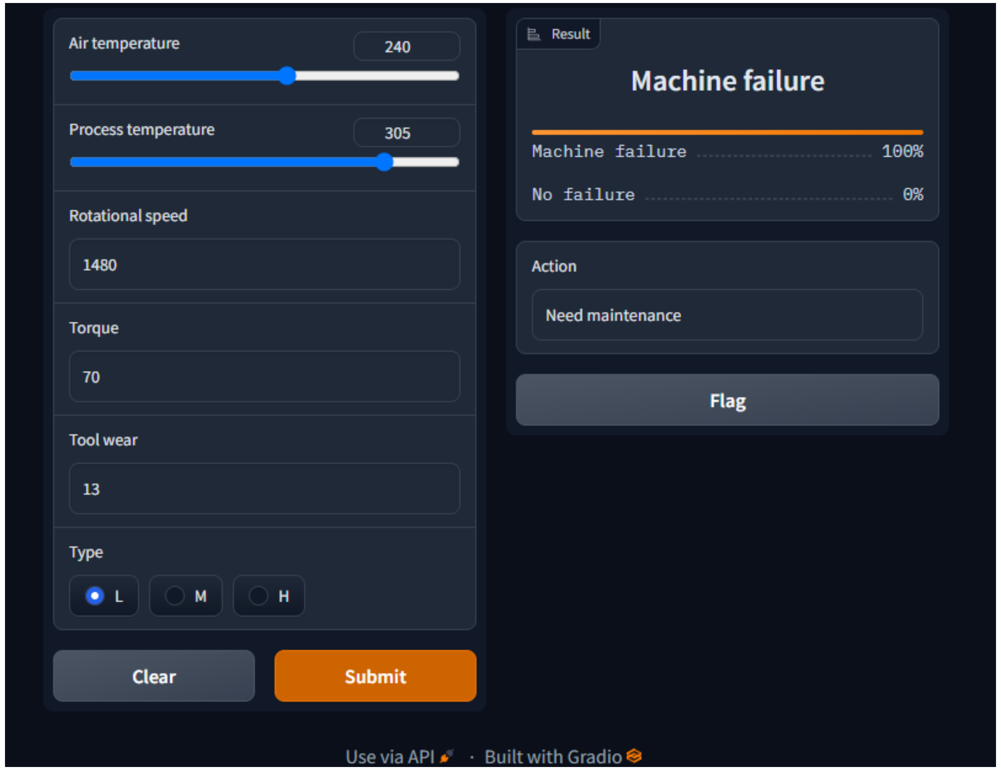
Predictive analytics

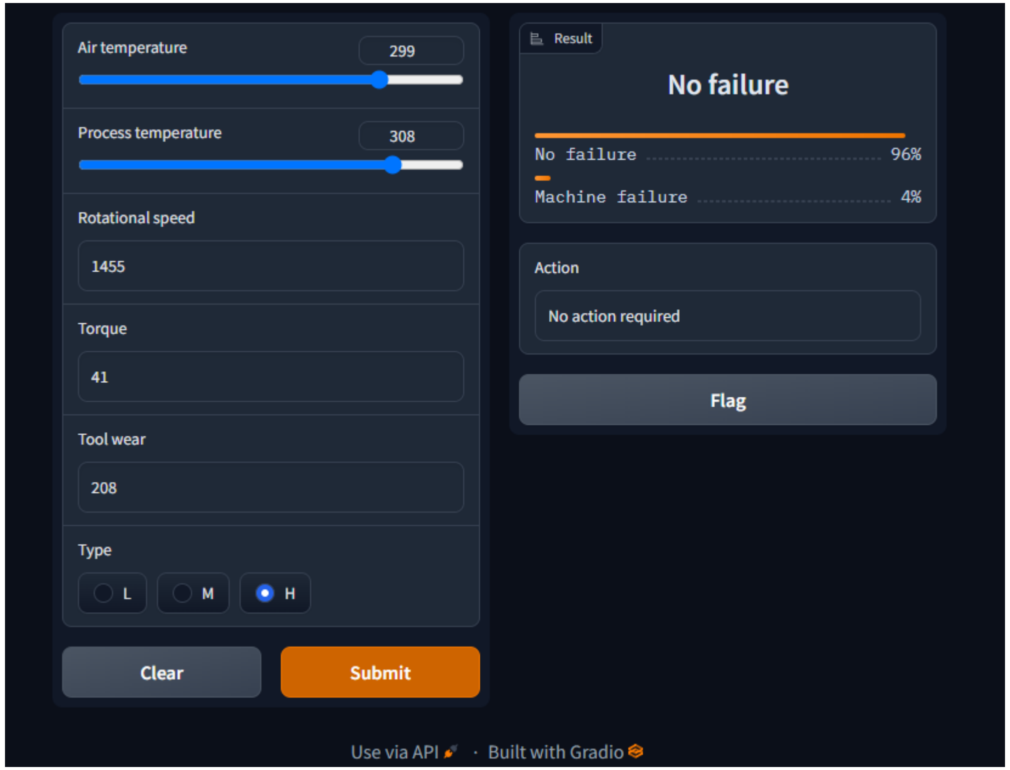
Models Evaluation

	Accuracy	Precision	Recall	F1-Score	Training time	Prediction time
Decision Tree	0.940426	0.969464	0.940426	0.952517	0.036945	0.002076
k-NN	0.973759	0.970333	0.973759	0.971756	0.019094	0.117287
Random Forest	0.983333	0.981517	0.983333	0.981835	1.202211	0.050093
Gradient Boosting	0.937589	0.975604	0.937589	0.952327	1.778598	0.006899
Gaussian Naive Bayes	0.757801	0.973338	0.757801	0.840162	0.007099	0.001722

Deployment

Demo the predictive machine learning model using **Gradio**





Conclusions

- Machine learning has the potential to significantly contribute to the domain of predictive maintenance.
- The data set is highly biased, which makes developing an accurate model challenging.
- We must collect sufficient data to create a reliable machine learning model.
- The Random Forest model outperforms the other models.