PREDICTIVE MAINTENANCE ANALYTICS FOR MANUFACTURING EQUIPMENT

Eyram Kueviakoe

Bellevue University
DSC 550: Data Mining

Dr. Brett Werner

May 29, 2024

Abstract

The goal of this project is to develop predictive maintenance models to anticipate equipment failures in a manufacturing company. Using data collected using sensors, we applied logistic regression and random forest classifiers to predict the failures and the types of failures. The models achieved high accuracy and they can help to reduce the downtime and maintenance costs.

This project includes data cleaning, feature engineering and model evaluation.

Introduction

In manufacturing industries, unplanned equipment failures can be very disruptive, and this can cause production delays, increased costs and safety hazards. Traditional maintenance approaches, often reactive, fail to address these issues efficiently. In this project, we will address the need for a proactive predictive maintenance approach to reduce unplanned downtime and improve equipment reliability.

Predictive maintenance in manufacturing is crucial because it helps prevent unexpected equipment failures that can lead to expensive downtime and repairs. By predicting when machines might break down, companies can schedule maintenance at convenient times. This will help to reduce disruptions and keep production running smoothly. This not only saves money but also ensures that the equipment lasts longer.

Predictive maintenance also improves safety for workers by keeping equipment in good condition and by reducing the risk of accidents.

Stakeholders will benefit from a factory where machines almost never break down unexpectedly and all maintenance is planned ahead with the safety of the employees in mind. Predictive maintenance makes this vision a reality by using sensor data to forecast equipment failures before they happen. This proactive approach can help to save money and repair costs. Scheduled maintenance means the production lines keep running smoothly, increasing efficiency and productivity.

Investing in predictive maintenance not only makes sense financially, but also well-maintained equipment is less likely to cause accidents and this will ensure a safer workplace for all employees.

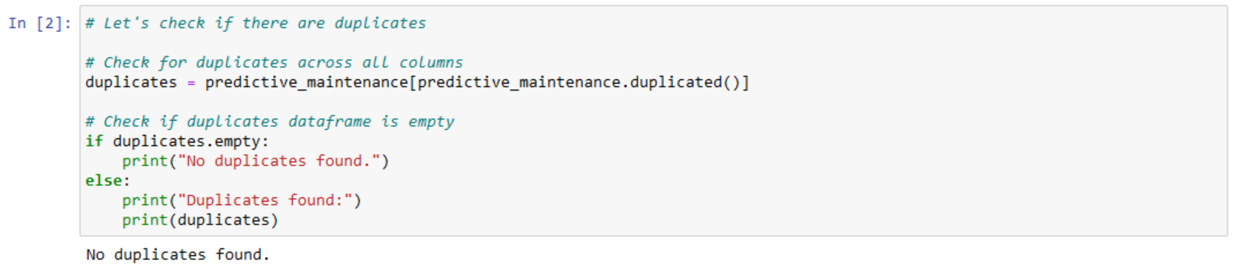
For this project, that data was obtained from Kaggle, a popular online platform that hosts thousands of datasets. The dataset we used includes various sensor readings and failure types from manufacturing equipment. It includes air temperature, process temperature, rotational speed, torque, and tool wear, and whether a failure occurred and the type of failure.

I. Data cleaning and preprocessing steps

Before building the models, the data was cleaned and pre-processed:

a. Checking for duplicates

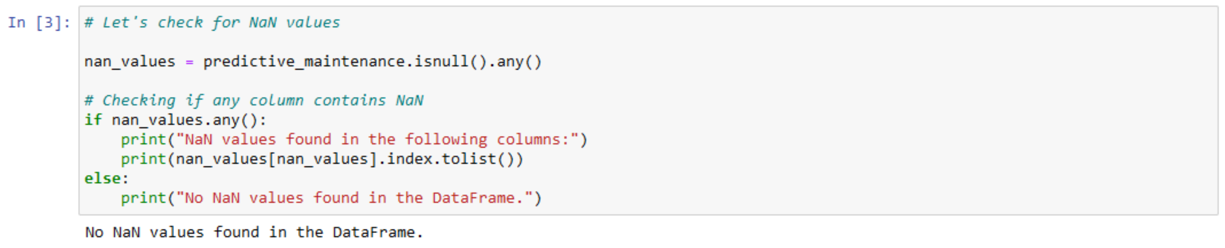
We check for duplicate rows in the dataset. This step is important to ensure that the analysis and the model are not biased by redundant data points.



```
In [2]: # Let's check if there are duplicates

        # Check for duplicates across all columns
        duplicates = predictive_maintenance[predictive_maintenance.duplicated()]

        # Check if duplicates dataframe is empty
        if duplicates.empty:
            print("No duplicates found.")
        else:
            print("Duplicates found:")
            print(duplicates)

        No duplicates found.
```

b. Checking for missing values

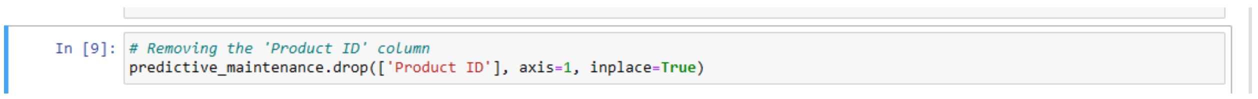
Checking for missing values helps to identify any gap in the dataset that needs to be addressed. Missing data can lead to a model and results that are inaccurate.



```
In [3]: # Let's check for NaN values

        nan_values = predictive_maintenance.isnull().any()

        # Checking if any column contains NaN
        if nan_values.any():
            print("NaN values found in the following columns:")
            print(nan_values[nan_values].index.tolist())
        else:
            print("No NaN values found in the DataFrame.")

        No NaN values found in the DataFrame.
```

c. Feature selection

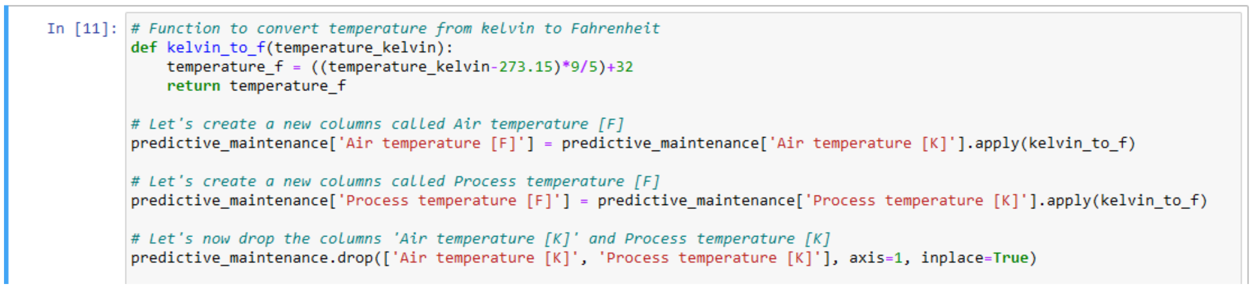
We dropped irrelevant columns such as ‘Product ID’ to focus on features that will contribute to the model predictions. Keeping columns that are irrelevant can introduce noise into the model.



```
In [9]: # Removing the 'Product ID' column
        predictive_maintenance.drop(['Product ID'], axis=1, inplace=True)
```

d. Feature Transformation

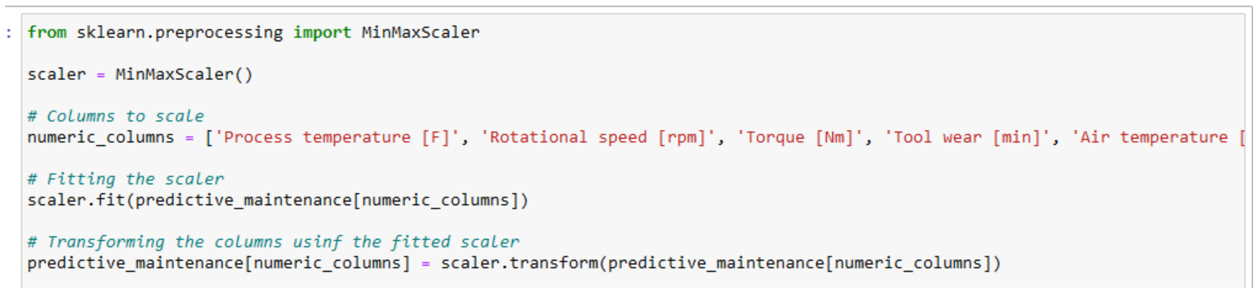
The temperature in the dataset was captured in kelvin. We converted the temperature to Fahrenheit to make it more interpretable.



```
In [11]: # Function to convert temperature from kelvin to Fahrenheit
         def kelvin_to_f(temperature_kelvin):
             temperature_f = ((temperature_kelvin-273.15)*9/5)+32
             return temperature_f

         # Let's create a new columns called Air temperature [F]
         predictive_maintenance['Air temperature [F]'] = predictive_maintenance['Air temperature [K]'].apply(kelvin_to_f)

         # Let's create a new columns called Process temperature [F]
         predictive_maintenance['Process temperature [F]'] = predictive_maintenance['Process temperature [K]'].apply(kelvin_to_f)

         # Let's now drop the columns 'Air temperature [K]' and Process temperature [K]
         predictive_maintenance.drop(['Air temperature [K]', 'Process temperature [K]'], axis=1, inplace=True)
```

e. Scaling Numerical Features

This step is important because it helps to ensure that all features are on the same scale.



```
: from sklearn.preprocessing import MinMaxScaler

  scaler = MinMaxScaler()

  # Columns to scale
  numeric_columns = ['Process temperature [F]', 'Rotational speed [rpm]', 'Torque [Nm]', 'Tool wear [min]', 'Air temperature [

  # Fitting the scaler
  scaler.fit(predictive_maintenance[numeric_columns])

  # Transforming the columns usinf the fitted scaler
  predictive_maintenance[numeric_columns] = scaler.transform(predictive_maintenance[numeric_columns])
```

f. Feature Engineering

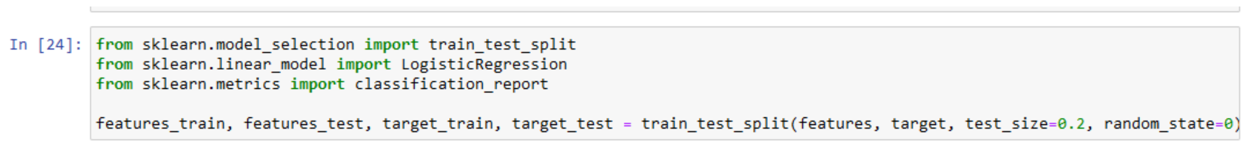
We created new features: Power consumption, temperature difference and temperature ratio. This steps is important because it helps to capture relationships between existing features.

g. Binary Classification Preparation

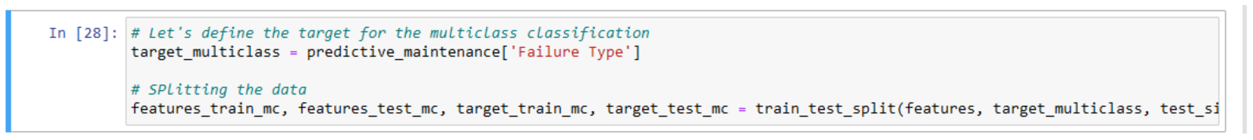
We added a binary column, "Failure". This helps to make a difference between failure and no failure, to address class imbalance and predict failures accurately.

h. Splitting the Data

Splitting the data into training and testing sets ensures that we can evaluate the model's performance on unseen data. This is an important step to prevent overfitting.



```
In [24]: from sklearn.model_selection import train_test_split
         from sklearn.linear_model import LogisticRegression
         from sklearn.metrics import classification_report

         features_train, features_test, target_train, target_test = train_test_split(features, target, test_size=0.2, random_state=0)
```

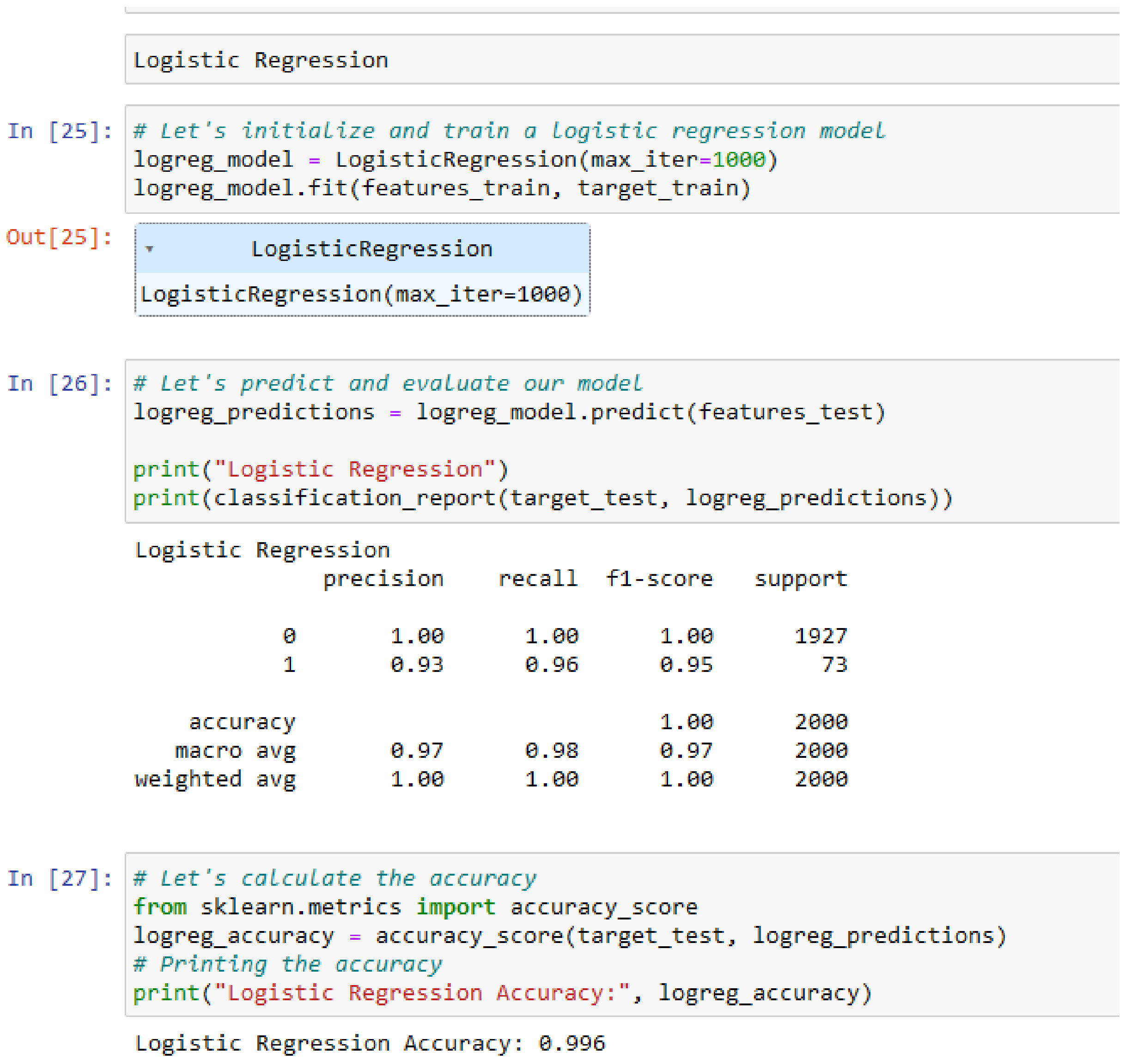


```
In [28]: # Let's define the target for the multiclass classification
         target_multiclass = predictive_maintenance['Failure Type']

         # Splitting the data
         features_train_mc, features_test_mc, target_train_mc, target_test_mc = train_test_split(features, target_multiclass, test_si
```

II. Model building and evaluation

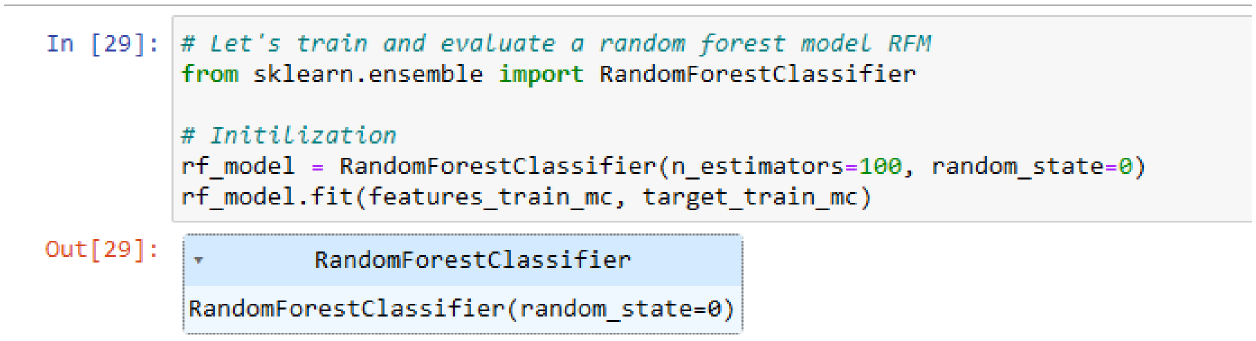
For this project, we chose two models: a logistic regression for binary classification and Random Forest.

A logistic regression for binary classification in the predictive maintenance is able to predict binary outcomes: Failure or no failure. It estimates the probability of the failure to occur based on the input features (equipment and air temperatures, RPM etc).

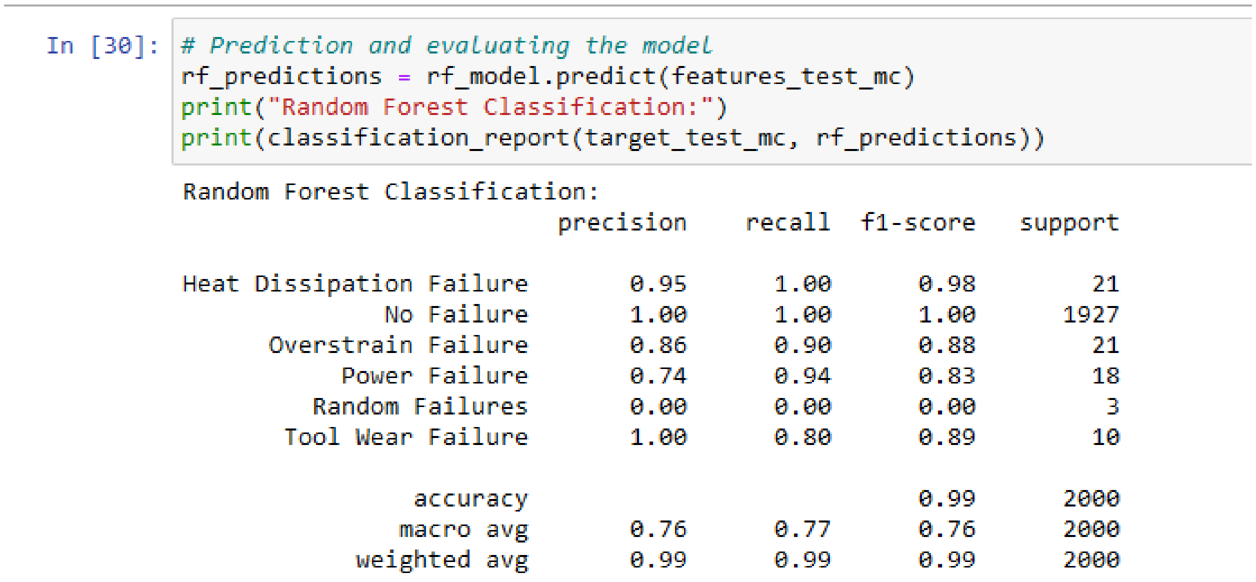


```
Logistic Regression

In [25]: # Let's initialize and train a Logistic regression model
         logreg_model = LogisticRegression(max_iter=1000)
         logreg_model.fit(features_train, target_train)

Out[25]:        LogisticRegression
         LogisticRegression(max_iter=1000)

In [26]: # Let's predict and evaluate our model
         logreg_predictions = logreg_model.predict(features_test)

         print("Logistic Regression")
         print(classification_report(target_test, logreg_predictions))

         Logistic Regression
                       precision    recall  f1-score   support

                    0       1.00      1.00      1.00      1927
                    1       0.93      0.96      0.95        73

             accuracy                           1.00      2000
            macro avg       0.97      0.98      0.97      2000
         weighted avg       1.00      1.00      1.00      2000

In [27]: # Let's calculate the accuracy
         from sklearn.metrics import accuracy_score
         logreg_accuracy = accuracy_score(target_test, logreg_predictions)
         # Printing the accuracy
         print("Logistic Regression Accuracy:", logreg_accuracy)

         Logistic Regression Accuracy: 0.996
```

We use Random Forest to predict the type of failure. It combines multiple decision trees to capture nonlinear patterns.



```
In [29]: # Let's train and evaluate a random forest model RFM
         from sklearn.ensemble import RandomForestClassifier

         # Initilization
         rf_model = RandomForestClassifier(n_estimators=100, random_state=0)
         rf_model.fit(features_train_mc, target_train_mc)

Out[29]: ▾       RandomForestClassifier
         RandomForestClassifier(random_state=0)
```



```
In [30]: # Prediction and evaluating the model
         rf_predictions = rf_model.predict(features_test_mc)
         print("Random Forest Classification:")
         print(classification_report(target_test_mc, rf_predictions))

Random Forest Classification:
                          precision    recall  f1-score   support

Heat Dissipation Failure       0.95      1.00      0.98        21
              No Failure       1.00      1.00      1.00      1927
     Overstrain Failure       0.86      0.90      0.88        21
          Power Failure       0.74      0.94      0.83        18
        Random Failures       0.00      0.00      0.00         3
       Tool Wear Failure       1.00      0.80      0.89        10

                accuracy                           0.99      2000
               macro avg       0.76      0.77      0.76      2000
            weighted avg       0.99      0.99      0.99      2000
```

We evaluated both models using classification reports to assess precision, recall, F1 score and accuracy.

The Logistic Regression model for binary classification (failure/no failure) achieved an accuracy of 99.6%. This means the model is efficient in predicting whether a failure will occur.

The Random Forest model for multiclass classification (failure types) achieved an accuracy of 99.35%. The model is able to accurately predict the type of failure.

III. Conclusion

The analysis and the model building in this predictive maintenance project will help in equipment failure prediction. Both models demonstrated high accuracy in predicting whether a failure will occur and categorizing the failure type. They are also indicating the factors that are contributing to equipment failures. This will guide proactive maintenance strategies.

However, the model will still need to be optimized and tested with real-time data to enhance its reliability before deployment.