Predictive Maintenance

Insights from following the pdf:

- Data consists of 10000 unique products (10000 unique product IDs)
- Features: Air Temperature, Process Temperature, Rotational Speed, Torque, Toolwear, Type (categorical)
- Target Variable → Failure Type: No Failure, Tool Wear Failure, Overstrain Failure, Power Failure, Heat Dissipation Failure

Failure_type	
NF	9670
HDF	106
PWF	80
OSF	78
TWF	42

- The dataset is extremely imbalanced. Therefore proposed methods: class weights, SMOTE
- Best Model: GradientBoostingClassifier

```
Fitting 3 folds for each of 30 candidates, totalling 90 fits
Best fine-tuned model parameters:
{'model__max_depth': 5, 'model__n_estimators': 400}

{'Accuracy': 0.9864729458917836,
 'Balanced Accuracy': 0.7172120708748616,
 'Macro Recall': 0.7172120708748616,
 'Macro Precision': 0.6969487444100447,
 'Macro F1': 0.7062532255779402,
 'F1 Scores per Class': array([0.92682927, 0.99302866, 0.82352941, 0.78787879, 0.       ])}
```

Diagnosing the low performance

- Running the best model parameters with GradientBoostingClassifier, noticed the following:

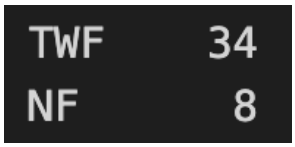
```
'F1 Scores per Class': {'HDF': 0.9047619047619048,
 'NF': 0.9925083957633687,
 'OSF': 0.7428571428571429,
 'PWF': 0.8235294117647058,
 'TWF': 0.0},
```

- It can be observed that the F1 scores of the classes are in the same order as its proportion in the dataset. Therefore the F1 score decreases as its representation reduces. This means that either
  - our methods (SMOTE and class weights) for mitigating the imbalance does not work.
  - our evaluation method is faulty
- Looking at the confusion matrix:

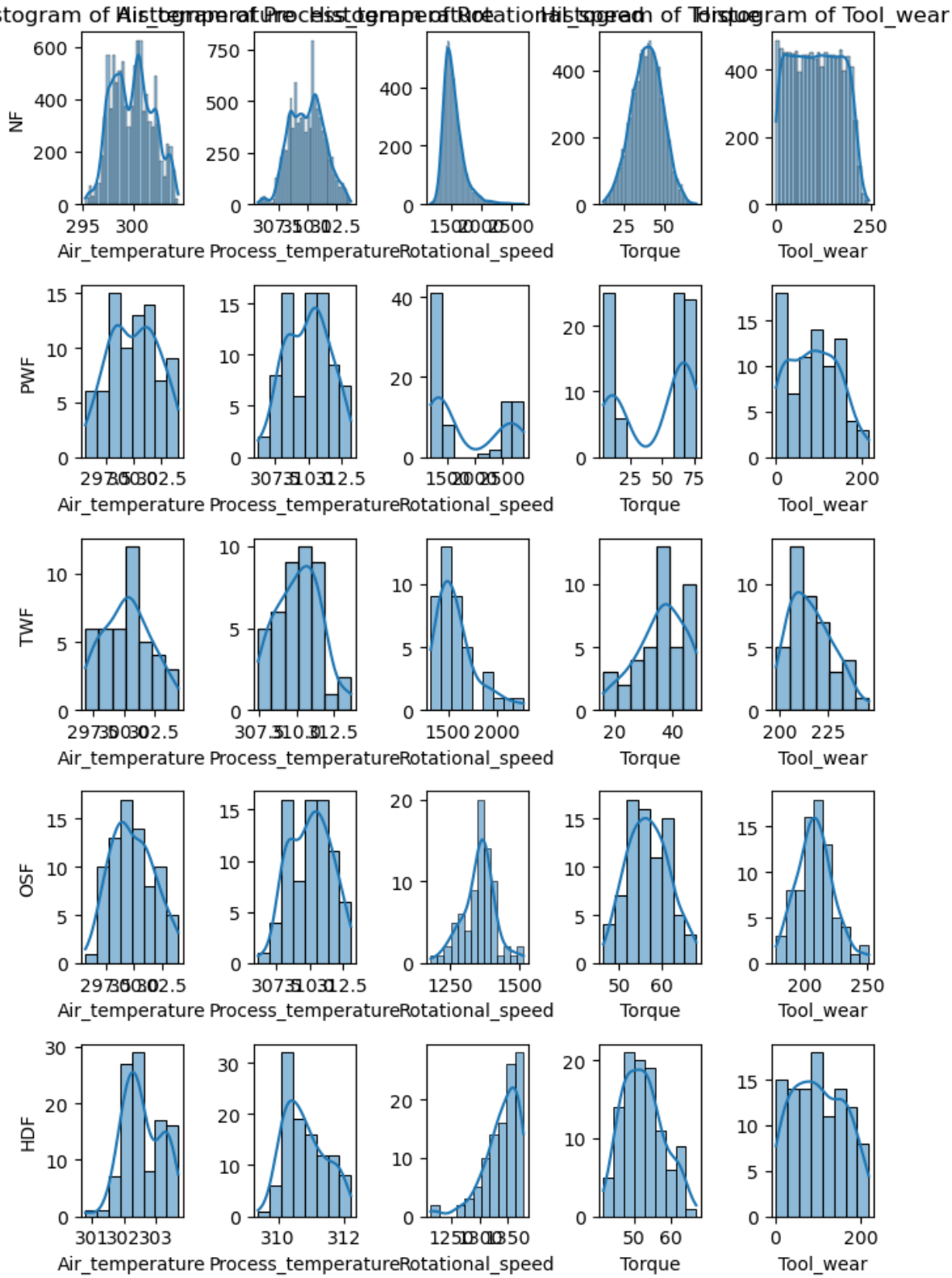
	Failure_type
NF	1935
HDF	21
PWF	16
OSF	16
TWF	8

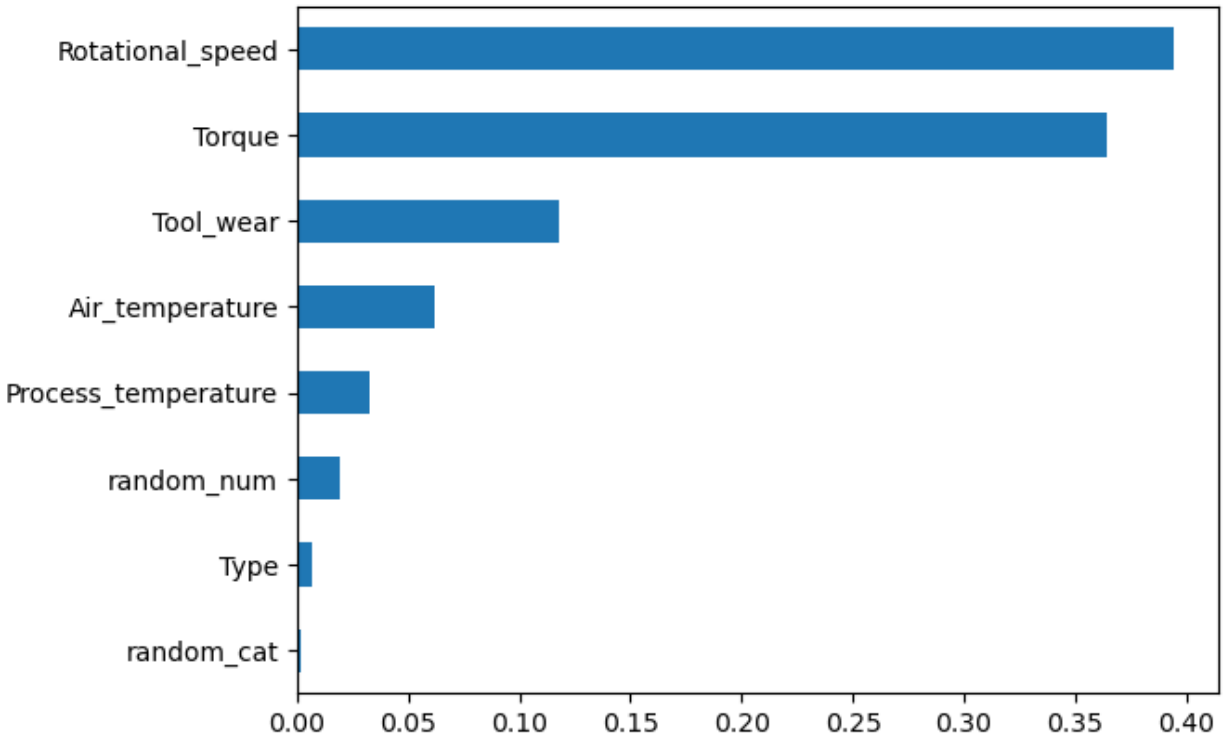
```
array([[  19,    2,    0,    0,    0],
       [   2, 1921,    6,    4,    2],
       [   0,    3,   13,    0,    0],
       [   0,    2,    0,   14,    0],
       [   0,    8,    0,    0,    0]])
```

    - Looking specifically at TWF, 8 TWF were predicted as NF, 2 NF were predicted as TWF
- To further investigate this, I fed the TWF samples to the trained model. It performed pretty well (High recall). However 8 samples were incorrectly classified. My guess (have to further investigate) is that all the training samples were correctly classified and test were not. Therefore it is memorising the data (overfitting). This is commonly seen in tree type methods (so not surprising)



- But why is this overfitting such a major issue mainly for TWF


Histogram of Air_temperature
Histogram of Process_temperature
Histogram of Rotational_speed
Histogram of Torque
Histogram of Tool_wear
NF
PWF
TWF
OSF
HDF
Air_temperature
Process_temperature
Rotational_speed
Torque
Tool_wear



- From these 2 figures, it can be seen that rotational speed is the most important feature in the dataset. In this feature, the distribution of instances are similar for NF and TWF. This is probably why it is getting confused and mis classifying TWF as NF (due to similar pattern in most discernable feature)
- Few things that might break my hypothesis:
  - Check if the 8 instances are the ones in the test set
  - Find another way to check if a model is overfitted
  - Is it really overfitting, or is it just the property of tree based models to correctly identify all train examples
  - Feauture importance in tree based methods are biased to numeric features with more variation

Neural nets Model

- used Multi layer perceptron classifier

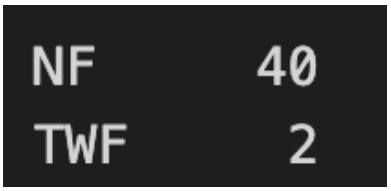
- Was not able to model tune

```
{'Accuracy': 0.9879759519038076,
 'Balanced Accuracy': 0.7204983388704319,
 'Macro Recall': 0.7204983388704319,
 'Macro Precision': 0.701796650935209,
 'Macro F1': 0.7109004400692067,
 'F1 Scores per Class': {'HDF': 0.8372093023255814,
  'NF': 0.9938080495356038,
  'OSF': 0.875,
  'PWF': 0.8484848484848485,
  'TWF': 0.0},
 'Recall Scores per Class': {'HDF': 0.8571428571428571,
  'NF': 0.9953488372093023,
  'OSF': 0.875,
  'PWF': 0.875,
  'TWF': 0.0},
 'Precision Scores per Class': {'HDF': 0.8181818181818182,
  'NF': 0.9922720247295209,
  'OSF': 0.875,
  'PWF': 0.8235294117647058,
  'TWF': 0.0}}
```

- Looking at confusion matrix:

```
array([[  21,    0,    0,    0,    0],
       [  10, 1890,    5,    6,   24],
       [   0,    3,   13,    0,    0],
       [   0,    1,    0,   15,    0],
       [   0,    6,    0,    0,    2]])
```

- For TWF: 6 incorrectly classified as NF
- Using all instances of TWF:



(investigate if same 2 are correct) → if this is the case then training did not work at all. Many of the instances that mustve been present during training are also being incorrectly identified. Further validated by the confusion matrix on X_train

```
array([[  72,   13,    0,    0,    0],
       [   6, 7725,    0,    2,    2],
       [   0,    6,   56,    0,    0],
       [   0,    6,    0,   58,    0],
       [   0,   32,    0,    0,    2]])
```

- Notice that 32 TWF samples in training were incorrectly classified. It is evident that during testing also its performance will be low
- Results after removing TWF:

```
{'Accuracy': 0.99194765978862 61,
 'Balanced Accuracy': 0.8733510919879844,
 'Macro Recall': 0.8733510919879844,
 'Macro Precision': 0.8943970204436712,
 'Macro F1': 0.8833414082687339,
 'F1 Scores per Class': {'HDF': 0.85,
  'NF': 0.9958656330749354,
  'OSF': 0.875,
  'PWF': 0.8125,
  'TWF': 0.0},
 'Recall Scores per Class': {'HDF': 0.8095238095238095,
  'NF': 0.9963805584281282,
  'OSF': 0.875,
  'PWF': 0.8125,
  'TWF': 0.0},
 'Precision Scores per Class': {'HDF': 0.8947368421052632,
  'NF': 0.9953512396694215,
  'OSF': 0.875,
  'PWF': 0.8125,
  'TWF': 0.0}}
```