PREDICTIVE MAINTENANCE

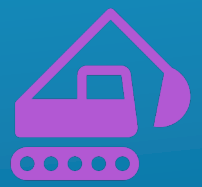
KAYLEIGH JAMES

1/13/2023

MOTIVATION

Machining and tools in industrial and manufacturing environments require high reliability. Machining failures directly impact revenue through lost production time.

For example, if DTP is running at 60 JPH then one minute of downtime results in a ~$50,000 loss in revenue.



In addition to revenue loss due to downtime, equipment failure results in cleanup costs, replacement/repair costs, collateral damage costs, and create safety issues.

Predictive maintenance can prevent these costs associated with failures and provide cost-saving over time-based, and routine based preventative maintenance.

DATASET

- The data in this analysis comes from a paper by Stephan Matzka for the 2020 Third International Conference on Artificial Intelligence for Industries (AI4I).
- "Since real predictive maintenance datasets are generally difficult to obtain and in particular difficult to publish, we present and provide a synthetic dataset that reflects real predictive maintenance encountered in industry to the best of our knowledge."
- The dataset contains 10,000 observations and 10 columns (8 are features and 2 are targets)

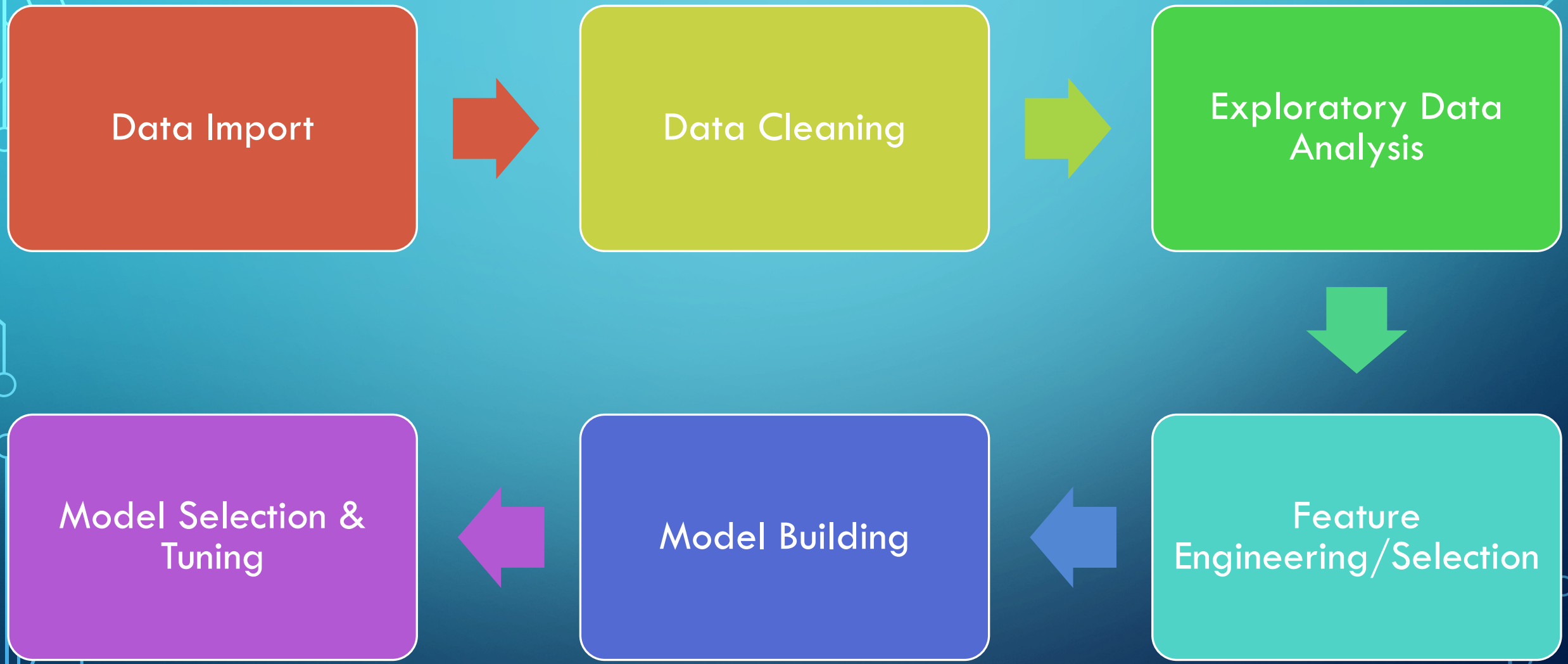
DATASET (CONT.)

- This dataset provides the opportunity for two separate classification problems:
  - Binary Classification:
    - Column labeled “Target”
    - 0 if the machinery was operational, 1 if the machinery failed
  - Multiclass Classification:
    - Column labeled ”Failure Type”
    - If the machinery was operational (e.g. Target = 0) the value will be “No Failure”
    - If the machinery failed (e.g. Target = 1) there a 5 possible failure types: Heat Dissipation Failure, Power Failure, Overstrain Failure, Tool Wear Failure, and Random Failure

DATASET (CONT.)

- There are 8 features we can use for both classification problems:
  - UID: unique identifier ranging from 1 to 10000
  - productID: 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
  - air temperature [K]: generated using a random walk process later normalized to a standard deviation of 2 K around 300 K
  - process temperature [K]: generated using a random walk process normalized to a standard deviation of 1 K, added to the air temperature plus 10 K.
  - rotational speed [rpm]: calculated from power of 2860 W, overlaid with a normally distributed noise
  - torque [Nm]: torque values are normally distributed around 40 Nm with an $\ddot{I}f = 10$ Nm and no negative values.
  - tool wear [min]: The quality variants H/M/L add 5/3/2 minutes of tool wear to the used tool in the process

PREDICTIVE ANALYTICS PROCESS FLOW



```
df.head(8)
```

	UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Target	Failure Type
0	1	M14860	M	298.1	308.6	1551	42.8	0	0	No Failure
1	2	L47181	L	298.2	308.7	1408	46.3	3	0	No Failure
2	3	L47182	L	298.1	308.5	1498	49.4	5	0	No Failure
3	4	L47183	L	298.2	308.6	1433	39.5	7	0	No Failure
4	5	L47184	L	298.2	308.7	1408	40.0	9	0	No Failure
5	6	M14865	M	298.1	308.6	1425	41.9	11	0	No Failure
6	7	L47186	L	298.1	308.6	1558	42.4	14	0	No Failure
7	8	L47187	L	298.1	308.6	1527	40.2	16	0	No Failure

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 10 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   UDI                      10000 non-null  int64  
 1   Product ID               10000 non-null  object 
 2   Type                     10000 non-null  object 
 3   Air temperature [K]      10000 non-null  float64
 4   Process temperature [K]  10000 non-null  float64
 5   Rotational speed [rpm]   10000 non-null  int64  
 6   Torque [Nm]              10000 non-null  float64
 7   Tool wear [min]          10000 non-null  int64  
 8   Target                   10000 non-null  int64  
 9   Failure Type             10000 non-null  object 
```

DATA IMPORT

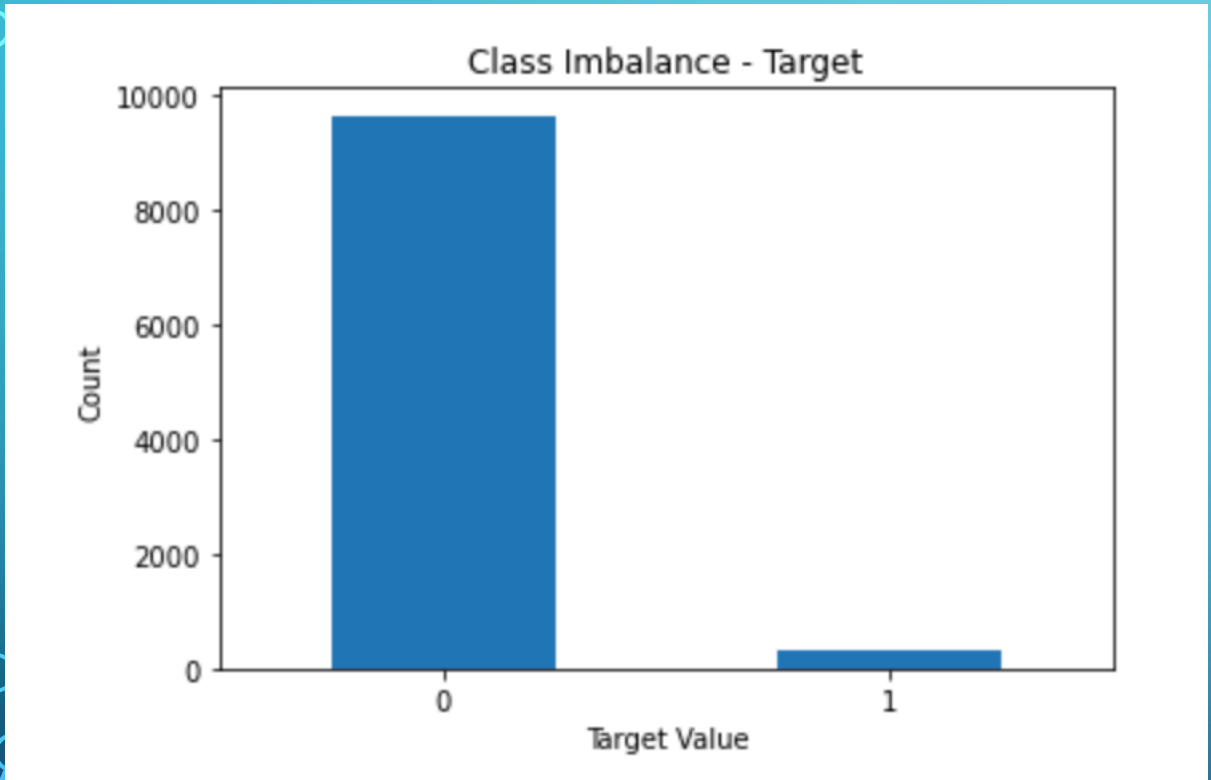
Library used: pandas

DATA CLEANING

- Checked for Null values in the data – there were none
- Checked for duplicates – there were none
- Dropped unnecessary columns – ProductID and UDI
- Data cleaning was minimal due to this being a synthetic dataset

Library used: pandas

EXPLORATORY DATA ANALYSIS



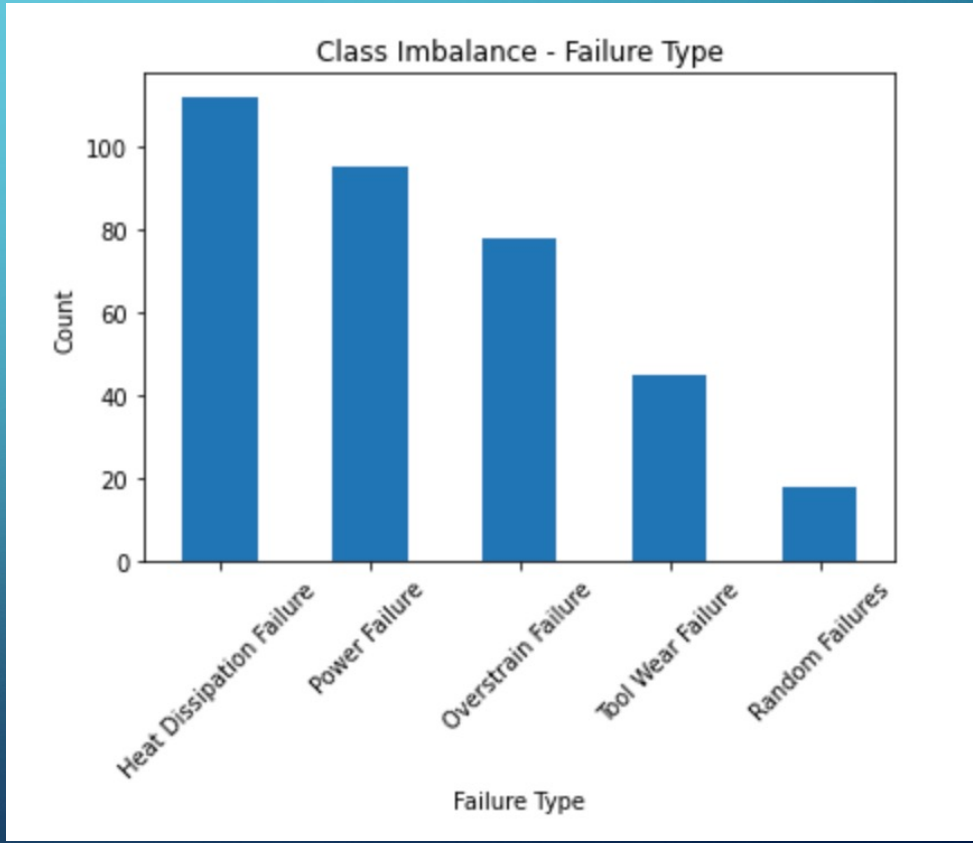
- For the binary classification problem the dataset is very imbalanced – this will have to be dealt with.
- Of the 10,000 observations only 339 had a failure.

Libraries used: matplotlib

EXPLORATORY DATA ANALYSIS

No Failure	9652
Heat Dissipation Failure	112
Power Failure	95
Overstrain Failure	78
Tool Wear Failure	45
Random Failures	18

- For the multiclass classification problem the data set was also imbalanced due to so few machine failures.
- Here is the distribution of failure types excluding “no failure”
- Once again, the classes might need to be balanced depending on model performance



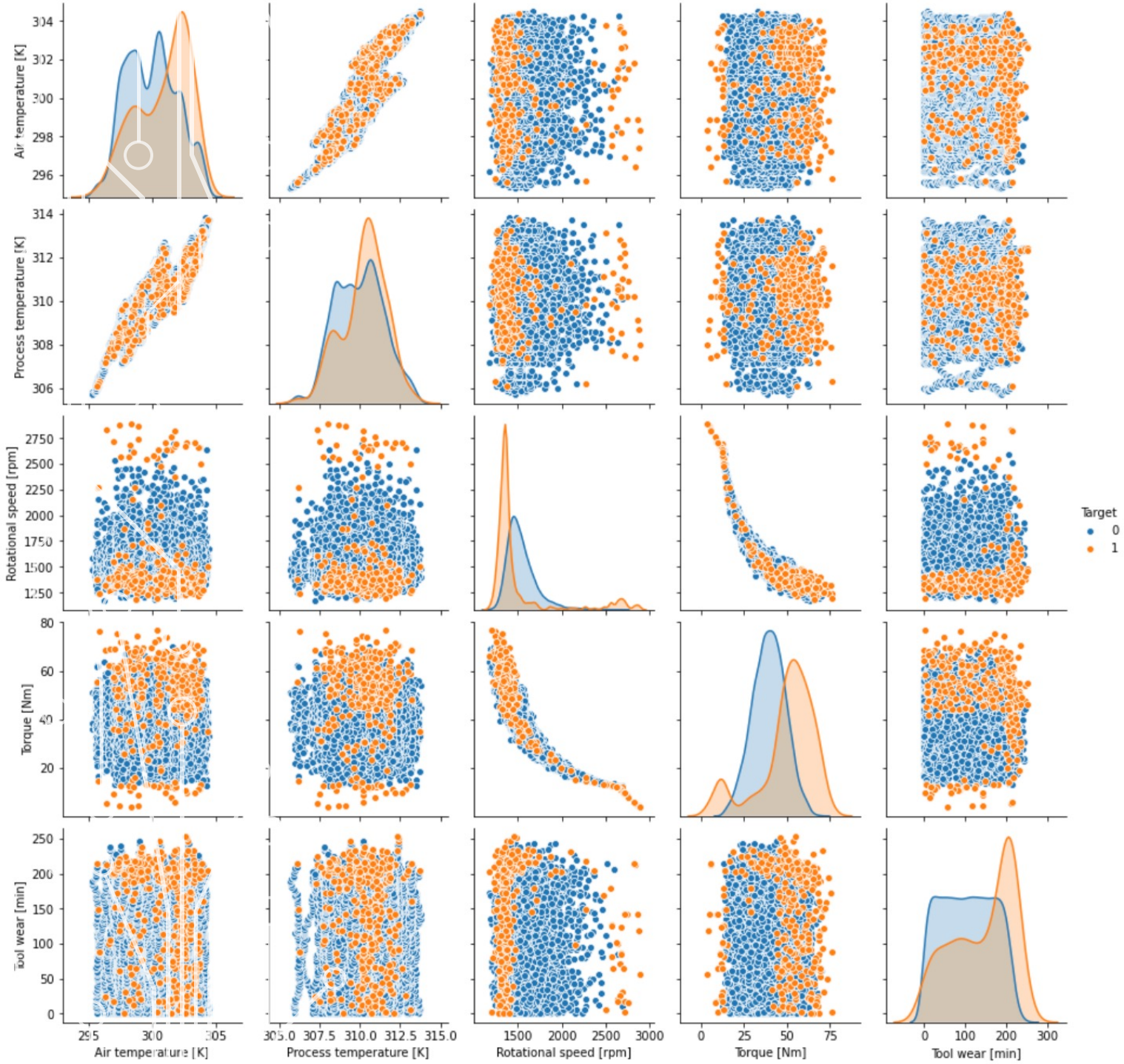
Libraries used: matplotlib and pandas

EXPLORATORY DATA ANALYSIS

	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000
mean	300.004930	310.005560	1538.776100	39.986910	107.951000
std	2.000259	1.483734	179.284096	9.968934	63.654147
min	295.300000	305.700000	1168.000000	3.800000	0.000000
25%	298.300000	308.800000	1423.000000	33.200000	53.000000
50%	300.100000	310.100000	1503.000000	40.100000	108.000000
75%	301.500000	311.100000	1612.000000	46.800000	162.000000
max	304.500000	313.800000	2886.000000	76.600000	253.000000

The numerical features have very different ranges from one another – because of this it is likely a good idea to perform feature scaling

Libraries used: pandas



EXPLORATORY DATA ANALYSIS

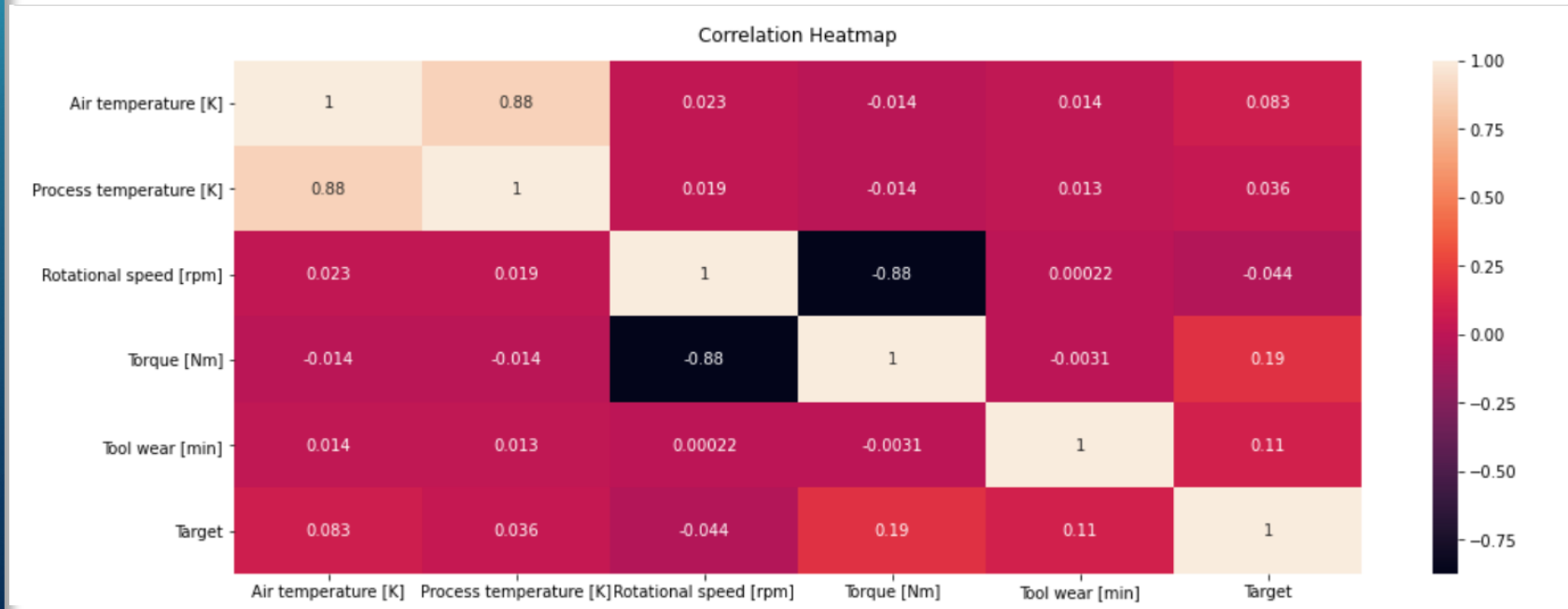
- Some features appear to be correlated – will likely need to perform some feature selection before model building
- Looks like some combinations of features do a pretty good job of separating the data (for the binary classification problem)

Libraries used: pandas

EXPLORATORY DATA ANALYSIS

- As suspected, there are some strongly correlated variables]
- Process Temperature and Air Temperature are positive correlated (this makes sense)
- Torque and Rotational Speed are strongly negatively correlated as well.

Libraries used: pandas, seaborn



BALANCING THE DATASET

- Before moving forward with feature selection and engineering I decided to balance the dataset.
- Of the 10,000 observations - 9,661 were 0/No Machine Failure and 339 were machine failures.
- I decided to upsample the minority class to 1,000.

FEATURE ENGINEERING

- For **feature scaling** I decided to normalize instead of standardize the features did not appear to have a Gaussian distribution. Normalizing can also help Gradient Descent and Distance based algorithms learn more quickly.
- "Type" is a categorical variable with three possible values. I used a **one-hot encoder** so that all predictive models will be able to work with the dataset.

FEATURE SELECTION

- Feature selection is motivated by several problems: Irrelevant/Redundant features, the curse of dimensionality, training time, and interpretability to name a few
- For this classification problem, there aren't many features so the curse of dimensionality and training time are not concerning. Given that we have seen some correlation between features I do think there are some redundant features to remove.

FEATURE SELECTION

- We had two sets of highly correlated features: Process Temperature and Air Temperature & Torque and Rotational Speed. Both had a correlation with an absolute value of 0.88
- I decided to drop Air Temperature and Rotational speed.
- Had this been a dataset with more features I would have used a more sophisticated method such as RFE, L1 Regularization, or tree-based feature selection

MODEL BUILDING – SPLITTING THE DATASET

- First we must split the data into 3 sets:
  - A training set – 70% of the data. Used to train the models.
  - A validation set – 15% of the data. Used to select the algorithm and tune hyperparameters.
  - A testing set – 15% of the data. Used to assess the tuned model.
- We stratify on the target variables so that we ensure we get the minority classes in both the validation and testing sets.

MODEL BUILDING – SPLITTING THE DATASET

```
#create holdout sets
X_mulit_train, X_multi_test, y_multi_train, y_multi_test = train_test_split(X_multi, y_multi,
                                                    stratify=y_multi,
                                                    test_size=0.30)
X_binary_train, X_binary_test, y_binary_train, y_binary_test = train_test_split(X_binary, y_binary,
                                                    stratify=y_binary,
                                                    test_size=0.30)

#split holdout sets into validation and training sets
X_mulit_val, X_multi_test, y_multi_val, y_multi_test = train_test_split(X_multi_test, y_multi_test,
                                                    stratify=y_multi_test,
                                                    test_size=0.50)
X_binary_val, X_binary_test, y_binary_val, y_binary_test = train_test_split(X_binary_test, y_binary_test,
                                                    stratify=y_binary_test,
                                                    test_size=0.50)
```

Libraries used: sklearn

MODEL BUILDING – CHOOSING ALGORITHMS

- I will be training the following classifiers:
  - Logistic Regression
  - Support Vector Machine (SVM) Linear and RBF
  - K-Nearest Neighbors (KNN)
  - Decision Tree

MODEL BUILDING – CHOOSING SCORING METRIC

$$precision = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \times precision \times recall}{precision + recall}$$

- Since we have an imbalanced dataset (even after upsampling the minority class), choosing accuracy would not be a good metric. The model could achieve a decent accuracy score by just assigning all of the observations in the holdout set to be in the majority class.
- Since we are focused on how well the model classifies the positive class (e.g. Machine Failure/Target=1) it makes sense to use the F-score which combines recall and precision.
- F1 score can be used for multiclass classification if you take an average. I used a weighted average because it takes label imbalance into account.

MODEL BUILDING - TRAINING

- Iterate over each classifier
- Calculate F1 score for each classifier and print the result

```
print("Binary Classification")
# iterate over classifiers
for name, clf in zip(names, classifiers):
    clf.fit(X_binary_train, y_binary_train)
    y_pred = clf.predict(X_binary_val)
    score = f1_score(y_binary_val, y_pred)
    print(name)
    print(score)

print("MultiClass Classification")
# iterate over classifiers
for name, clf in zip(names, classifiers):
    clf.fit(X_multi_train, y_multi_train)
    y_pred = clf.predict(X_multi_val)
    score = f1_score(y_multi_val, y_pred, average='weighted')
    print(name)
    print(score)
```

MODEL SELECTION – RESULTS FROM TRAINING

F1 scores come from using the model on the validation set

Algorithm	Binary Classification F1	Multiclass Classification F1
K-Nearest Neighbors	0.66	0.93
Linear SVM	0	0.90
RBF SVM	0.90	0.98
Decision Tree	0.63	0.92
Logistic Regression	0.25	0.90

HYPERPARAMETER TUNING

- Grid Search with Cross Validation performs an exhaustive search over specified parameter values for an estimator.
- Best Parameters for Binary Classification:

```
Best Parameters: {'C': 10, 'gamma': 1, 'kernel': 'rbf'}
```

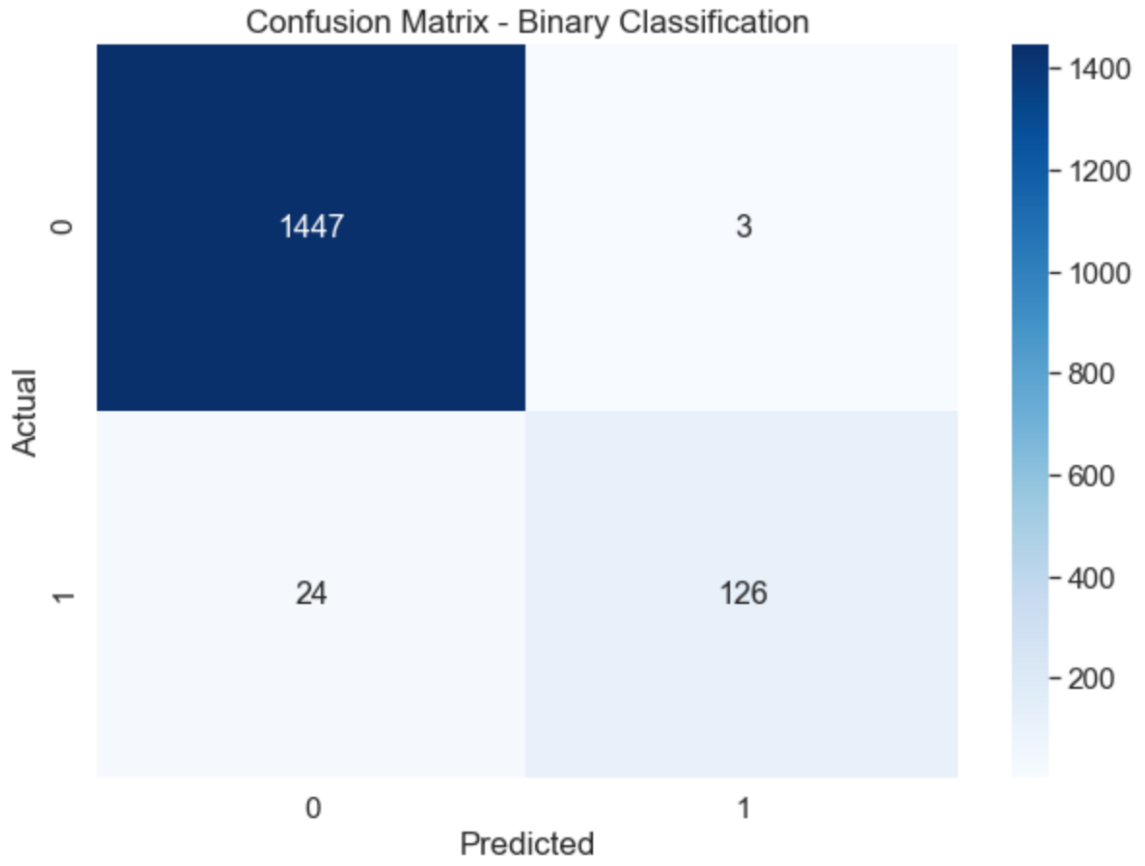
- Best Parameters for Multiclass Classification:

```
Best Parameters: {'C': 10, 'gamma': 1, 'kernel': 'rbf'}
```

FINAL MODEL EVALUATION – BINARY

Binary Classification w/ Best RBF SVM

	precision	recall	f1-score	support
0	0.98	1.00	0.99	1450
1	0.98	0.84	0.90	150
accuracy			0.98	1600
macro avg	0.98	0.92	0.95	1600
weighted avg	0.98	0.98	0.98	1600



FINAL MODEL EVALUATION – MULTICLASS

Multiclass Classification w/ Best RBF SVM

	precision	recall	f1-score	support
Heat Dissipation Failure	1.00	0.86	0.92	50
No Failure	0.98	1.00	0.99	1451
Overstrain Failure	1.00	0.87	0.93	38
Power Failure	1.00	0.83	0.91	41
Random Failures	0.00	0.00	0.00	2
Tool Wear Failure	0.87	0.72	0.79	18
accuracy			0.98	1600
macro avg	0.81	0.71	0.76	1600
weighted avg	0.98	0.98	0.98	1600

CONCLUSIONS

- We can see that with this dataset that you can generate decently reliable models for predicting failure given a few measurements.
- I believe the next steps would be:
  - Instead of just predicting failure - predict **when** to perform preventative maintenance so failures are avoided all together.
  - This would require streaming data as opposed to offline learning as well as time series analysis.