AIML CA1
Classification

01

Title:

Predicting Machine Failure

BY KALEB NIM P2100829 DAAA/2A/02

Table of Contents

PROJECT OBJECTIVE

Using Machine Learning classification model to solve Binary Class classification problem:

Predict the machine state of the Machine,
i.e machine will fail or machine remains normal functional based on machine attributes

OUTPUT VARIABLE

The output variable machine status is a **two-class label** - Normal or Failure.

Normal (0) is defined as: Machine will continue to function normally with no malfunction;
Whereas
Failure (1) is defined as: Machine will breakdown and needs to undergo maintenance

DATA ATTRIBUTES

Number of Instances: 20k

Number of Attributes: 9

Missing values are denoted as: NaN

DATA DICTIONARY

Column	Description
Unique ID	Unique identifier ranging from 1 to 20000
Product ID	The serial number of product
Quality	Consist of letter L, M and H for low, medium and high quality
Ambient T	Environment temperature in degree celsius
Process T	Machine temperature
Rotation Speed	Rotational speed of machine when running
Torque	Measure of the turning force
Tool Wear	Tool wear time estimated for the machine
Machine Status	**Labels indicate machine failure or not, 1 means failure, 0 means normal.**

02

Exploratory Data Analysis(EDA)

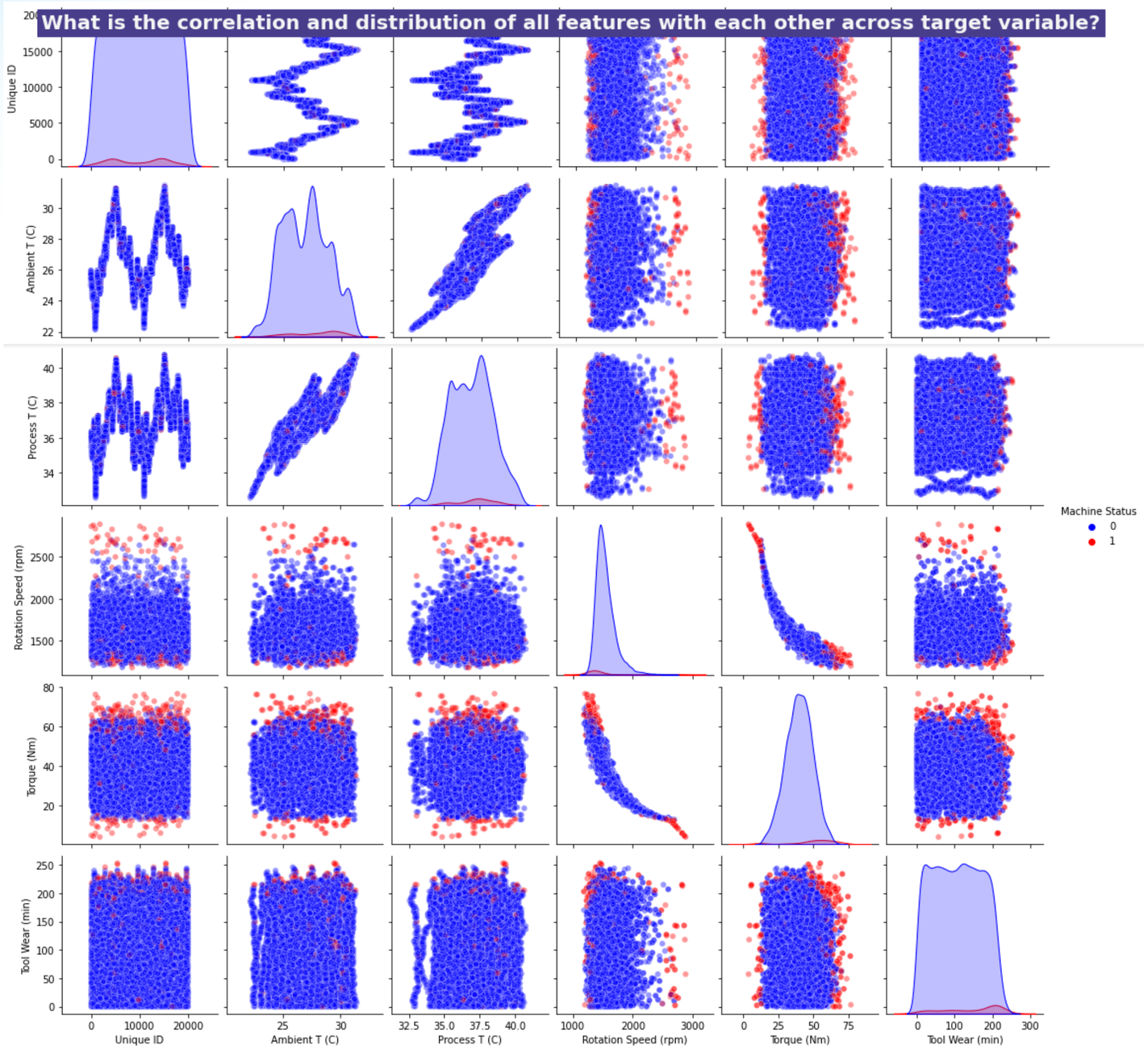
1.Uni-Variate Analysis across Machine status

2.Bi-Variate Analysis

Correlation Matrix

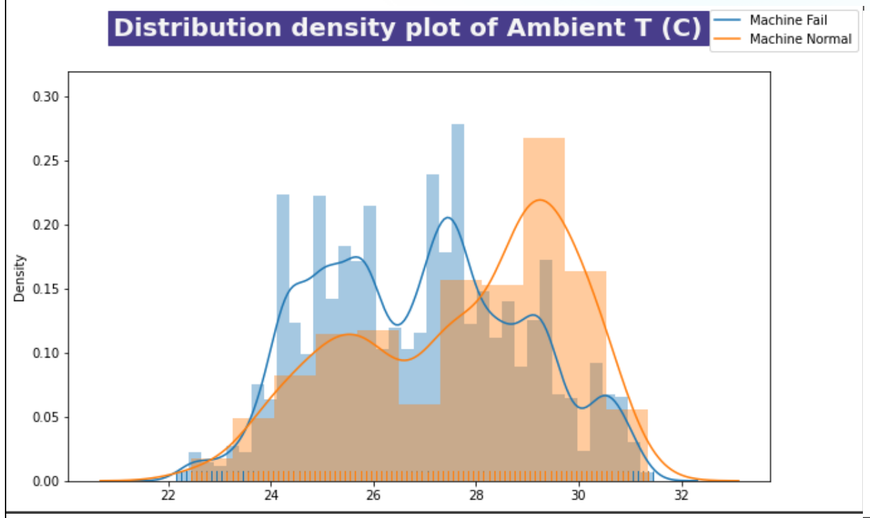
3.Target Distribution

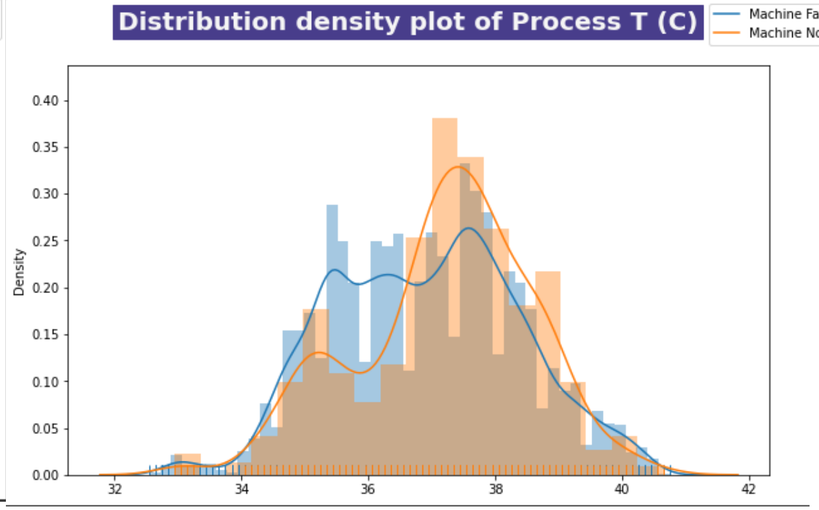
EDA

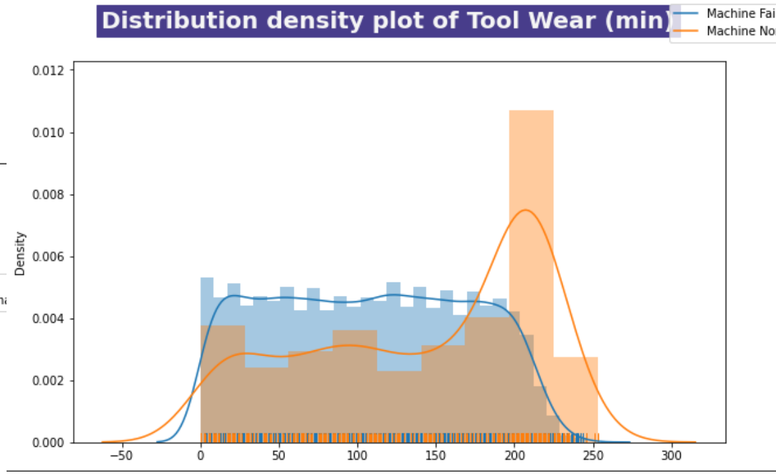

What is the correlation and distribution of all features with each other across target variable?
Unique ID
Ambient T (C)
Process T (C)
Rotation Speed (rpm)
Torque (Nm)
Tool Wear (min)
Machine Status
0
1

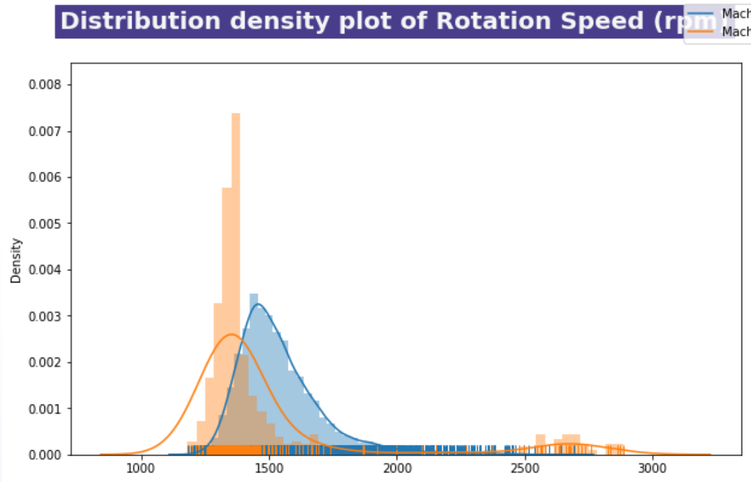
EDA

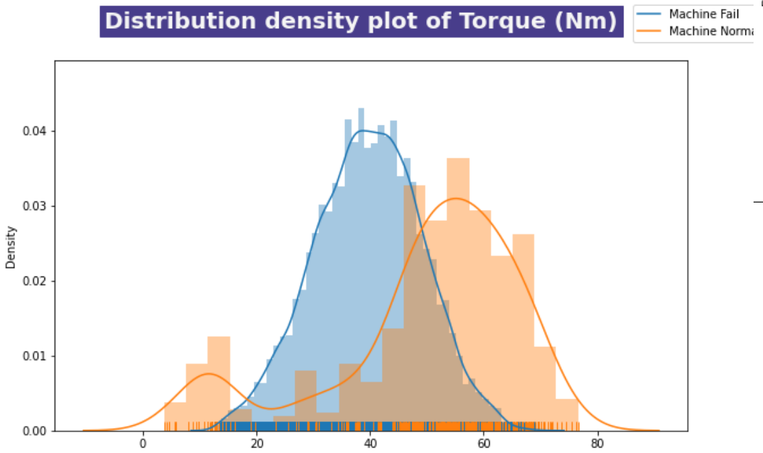
Uni-Variate Analysis across Machine status











EDA
Phik correlation metric:

Berif intro to Phik

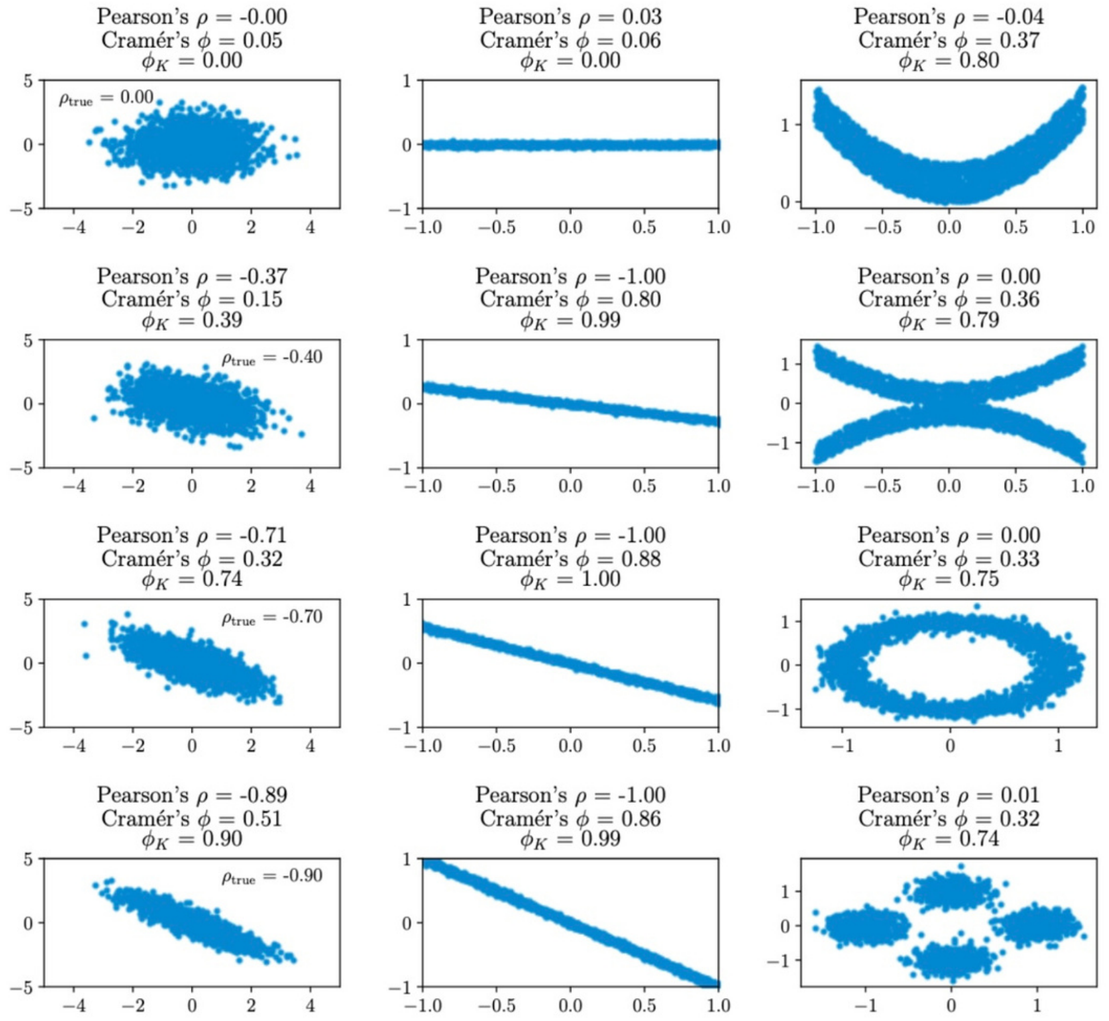
Phik correlation (ϕk) is the latest relatively new correlation metric that is based on several refinements to Pearson's χ2 (chi-squared) contingency test

Unfortunately there's isn't a closed forum formula . Therefore let see a visual representation of how Phik correlation works

Usage in EDA

We will be using Phik correlation in Bi-variation analysis: Check high correlation between two variable

Check highly correlated feature to target feature Price



source

EDA

Bi-variate Analysis

- **Correlation Metric**

  Phik correlation (ϕk)

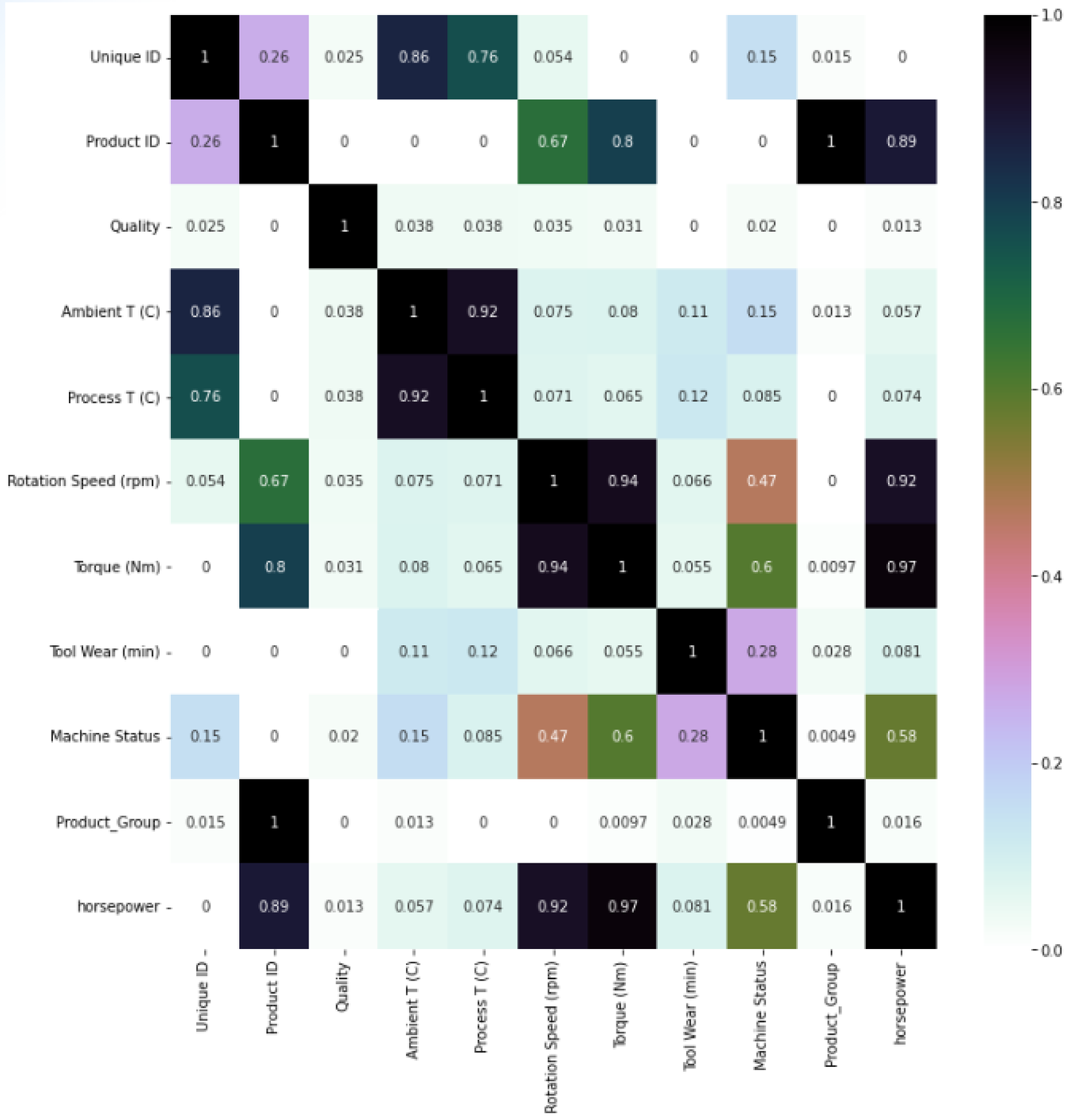
- **Greatest Correlation with Machine Status**

1. Torque (Nm) ----------> 0.6ϕk
2. Rotation Speed (rpm) --> 0.47ϕk
3. Tool Wear (min) ---------->0.28ϕk

- **Correlated Features**

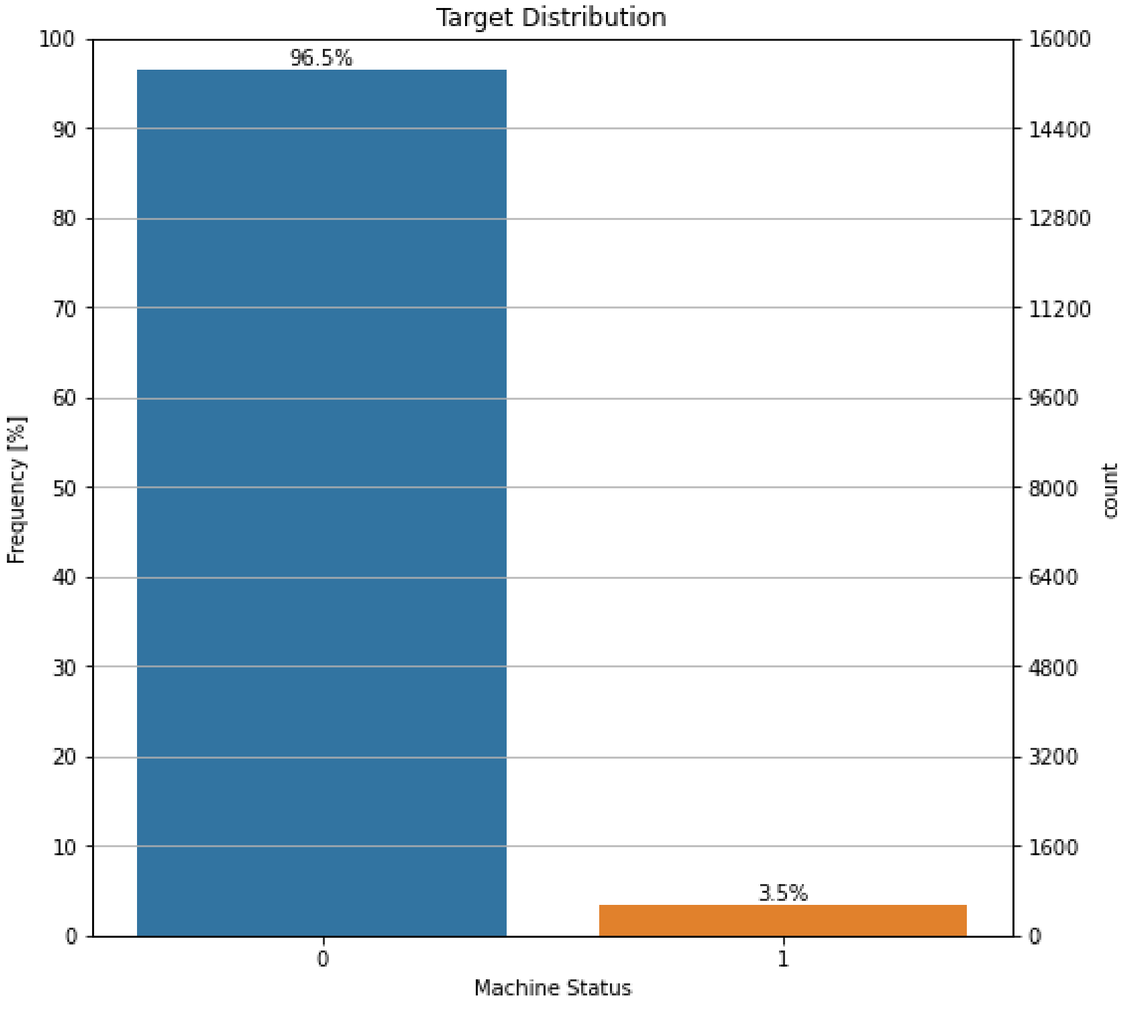
1. Ambient T (C) and Process T (C) is at 0.92ϕk

2. Torque (Nm) and Rotation Speed (rpm) at 0.93ϕk



EDA

Distribution of Taget Matrix



Extremely **unbalanced distribution** of observations

- Normal: 96.5%
- Failure: 3.5%

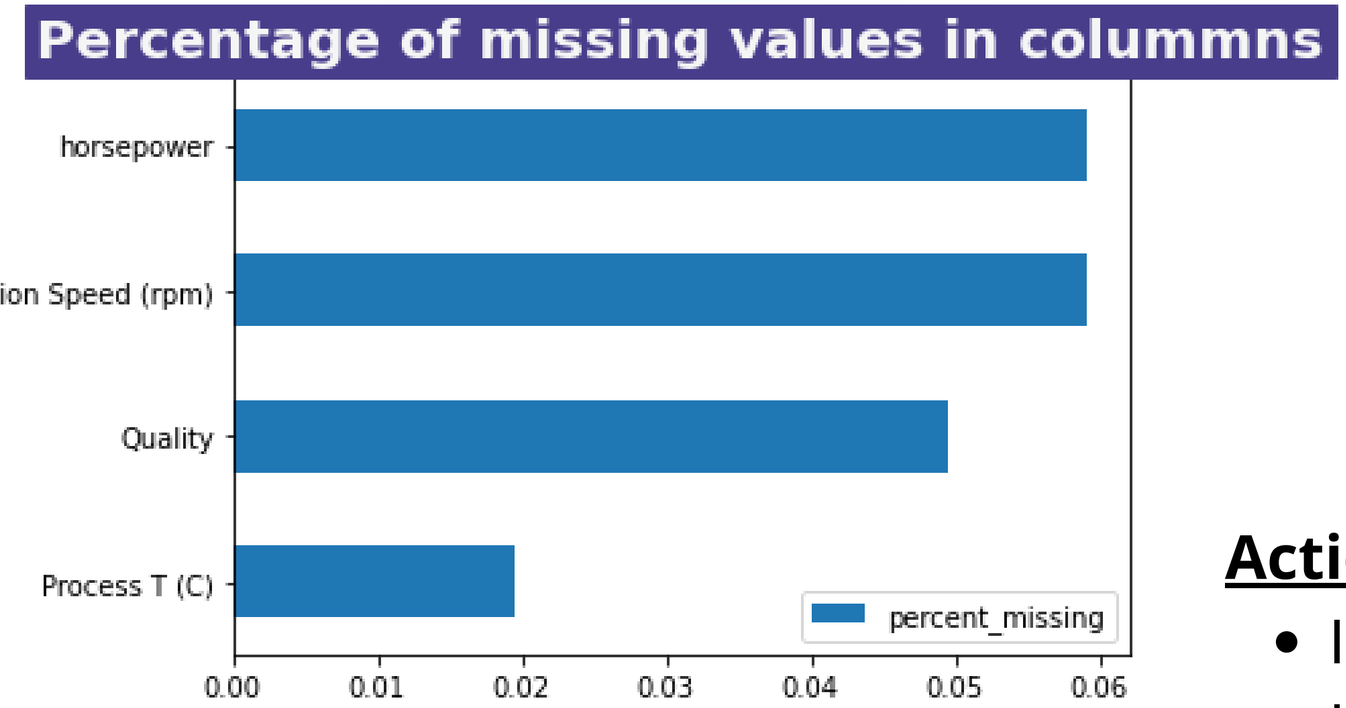
Conclusion: Imbalanced Dataset

Actions :

- Use **f1-beta** as the evaluation metric
- Data augmentation
  - **SMOTE**

EDA

Null Value distribution



Figures:

- horsepower 945count
- Rotation Speed (rpm) 945count
- Quality 792count
- Process T (C) 313 count

Actions :

- Impute with Central Tendency
- Impute with Advanced Algorithm in SKLearn(IterativeImputer, KNNImputer)

note: Housepower calculated from Rotation speed hence same No. null values

03

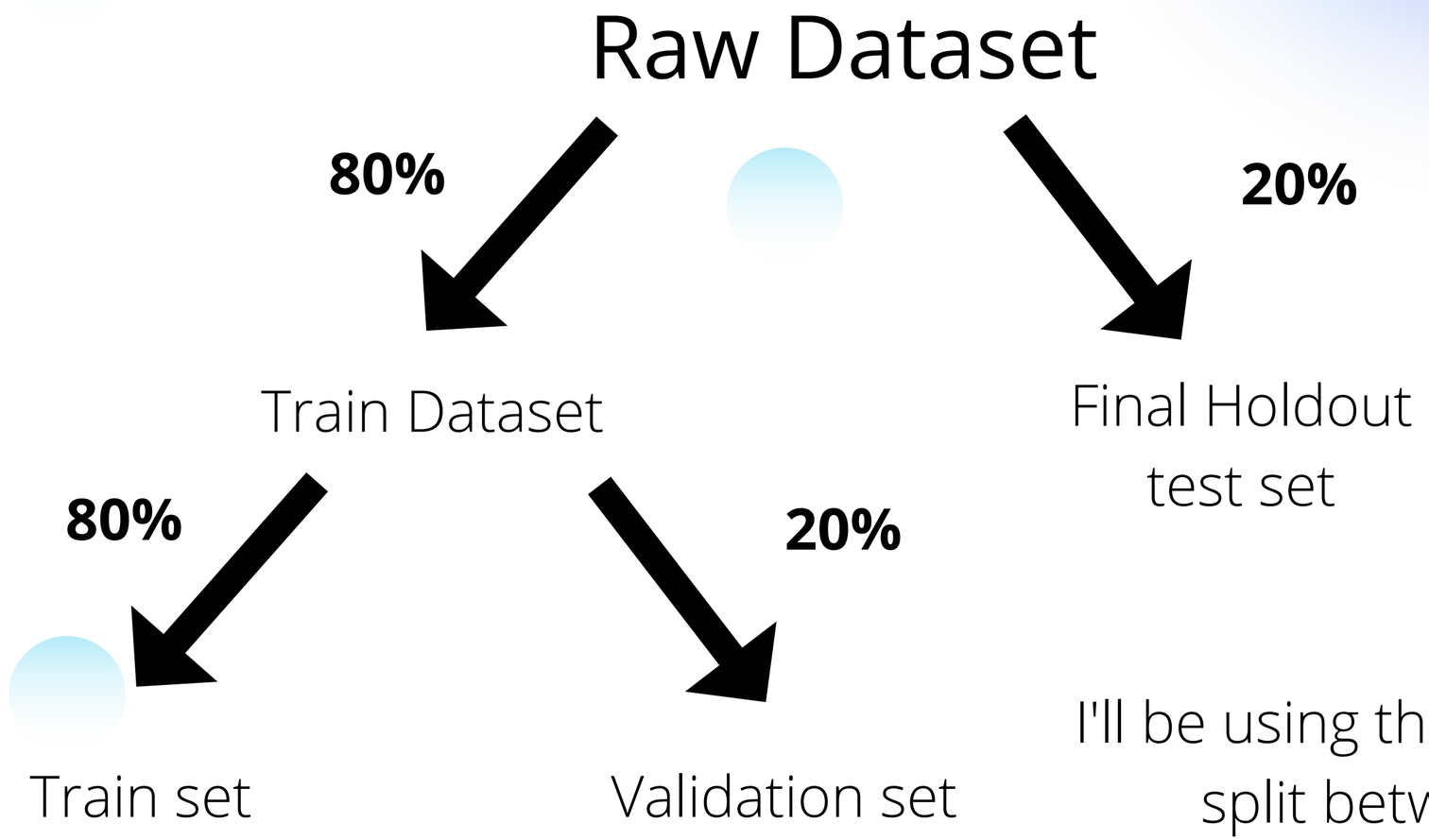
Feature Processing

List of techniques applied to dataset

- **Data Partition**
- **Imputation**
- **Feature Scaling**
- **Categorical Encoding**

Feature Processing

Data Partitioning



I'll be using the conventional **64-16-20** split between the 3 datasets

Feature Processing

Imputation

IterativeImputer

For numeric features

- Rotation Speed (rpm)
- Process T (C)

SimpleImputer

Impute with most frequent value

- Quality

Feature Processing

Categorical Encoding

Label Encoder

Quality

- 'L'--> 0
- 'M' --> 1
- 'H' --> 2

Why Label Encode:

Since Quality is a Ordinal feature, we want the model to derive a correlation between the Quality number increases and machine failure prediction

Feature Processing

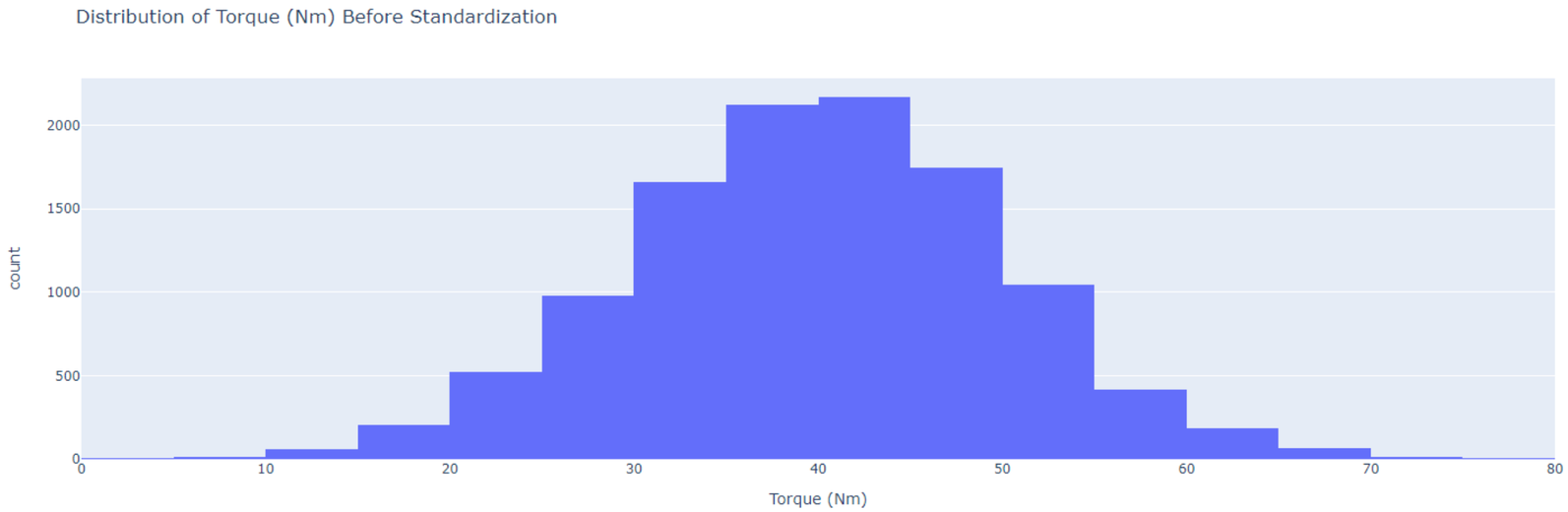
Feature Scaling

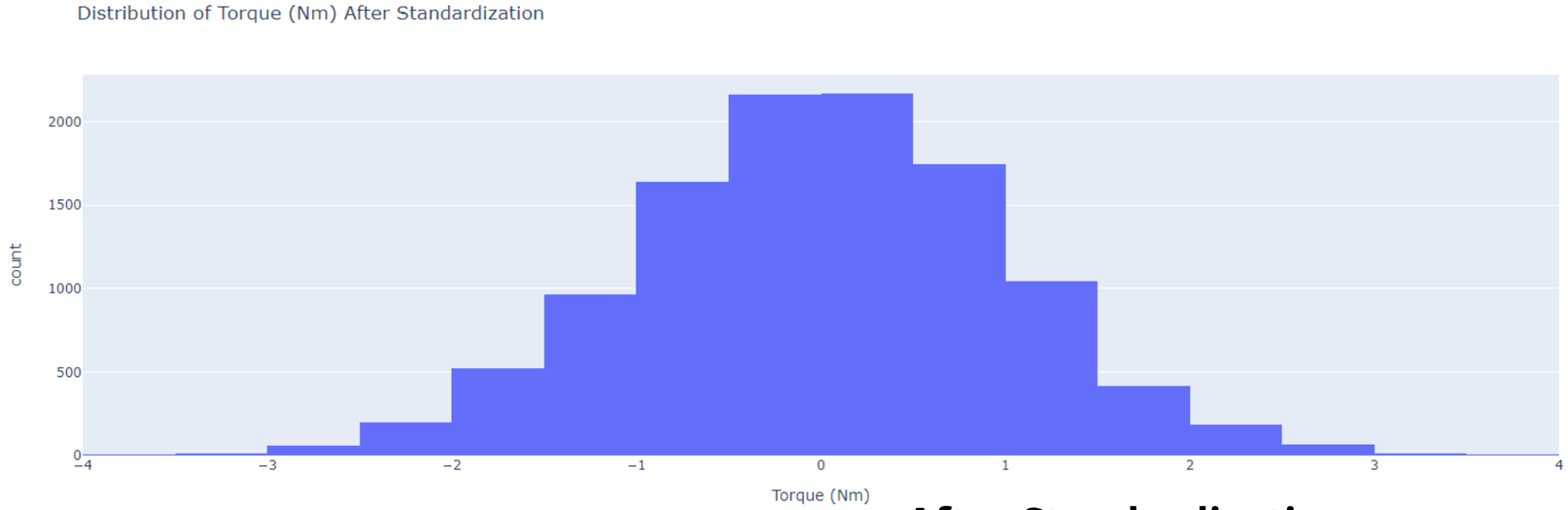
Standardscalar

Translating the all numerical features: mean value of zero and a standard deviation of one.

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

Example:





After Standardization,

- Mean value has change from 40 to 0
- Range from (10 --> 70) to (-3 --> 3)

04

Modelling

- **Selecting evaluation metrics**
- **Evalution Method**

MODELLING

SELECTING EVALUATION METRICS

Confusion Matrics for Machine failure

	Actually: Normal	Actually: Failure
Predicted: Normal	True Positives (TPs)	False Positives (FPs)
Predicted: Failure	False Negatives (FNs)	True Negatives (TPs)

fβ-score as our main evaluation metric as it gives us the flexiblilty to assign weights to prescision/recall. To greater penalize FP, assign beta=0.5 which will weigh precision 2x heavier than recall.

Note: There isn't a scoring key for fβ-score thus f1-score will be used instead for comparism in cv training and learning curve

04

Modelling

Evalution Method

- Cross validate using KFold each model on training set, and keeps a history of the model performance.
- Scoring on Validation set
- Plot learning curve of each model

```python
def quick_evalutaion(models, X_train, X_test, y_train, y_test,metrics=["precision","recall","f1","roc_auc"],curve = False, cv = KFold(n_splits=5,shuffle=True,random_state=1)):
    hist = {}

    for idx, model in (enumerate(models)):
        try:
            clf = model(random_state=42,class_weight='balanced') # Setting random_state for certain model + set class rate to balanced (imbalanced dataset )
        except:
            clf = model()
        clf.fit(X_train, y_train)
        test_prediction = clf.predict(X_test) # Testing on validation dataset

        # Test dataset
        f1_test = f1_score(y_test, test_prediction,zero_division=1)
        f1_beta = fbeta_score(y_test, test_prediction,zero_division=1,beta= 0.5) # β = 2, weighs Precision higher than Recall.
        test_recall = recall_score(y_test,test_prediction)
        test_prescision = precision_score(y_test,test_prediction)
        test_roc_auc_score = roc_auc_score(y_test,test_prediction)

        # 5-Fold CV
        cv_hist = cross_validate(clf, X_train, y_train, scoring=metrics,verbose=1)

        # Record down the performance
        hist[model.__name__] = dict(
            # train_acc = acc_train,
            cv_precision = cv_hist['test_precision'].mean(),
            prescision_test_score = test_prescision,
            cv_recall = cv_hist['test_recall'].mean(),
            recall_test_score = test_recall,
            cv_auc = cv_hist['test_roc_auc'].mean(),
            auc_score = test_roc_auc_score,
            cv_f1_score = cv_hist['test_f1'].mean(),
            f1_test_score = f1_test,
            f1_beta_test_score = f1_beta,

        )

        # Plotting the learning Curve of each Model using: F1_score
        if curve:
          fig, ax = plt.subplots(figsize=(10, 8))
          train_sizes = np.linspace(.1, 1.0, 10)
          train_sizes, train_scores, test_scores = learning_curve(clf, X_train, y_train, cv = cv, n_jobs = -1, train_sizes = train_sizes, scoring="f1")
          scores = pd.DataFrame({
          "Train Sizes" : np.tile(train_sizes, train_scores.shape[1]),
          "Train Scores" : train_scores.flatten(),
          "Test Scores" : test_scores.flatten()
          }).melt(value_vars=["Train Scores", "Test Scores"], var_name="Score Type", value_name="Scores", id_vars=["Train Sizes"])
          # print(f"THis is train_sizes:\n{train_sizes}\n This is train_scores:\n{train_scores}\n This is test_scores:{test_scores}")
          sns.lineplot(data=scores, x="Train Sizes", y="Scores", hue="Score Type", ax = ax,palette=['#3DD5E2', '#A045B5'])
          ax.set_title(f"Learning Curve of {model}")
          ax.set_ylabel("f1_score")
          ax.set_xlabel("Train Sizes")
          plt.show()

    results = pd.DataFrame(hist).T
    return results
```

05

Model Selection

Selecting Final model based on:

- **Variance/bias**
- **Interpretability**
- **Evaluation Metric performance**
  - **Chosen metric: fβ-score**

MODEL SELECTION

RESULTS

	cv_precision	prescision_test_score	cv_recall	recall_test_score	cv_auc	auc_score	cv_f1_score	f1_test_score	f1_beta_test_score
DummyClassifier	0.000000	0.000000	0.000000	0.000000	0.500000	0.500000	0.000000	0.000000	0.000000
LogisticRegression	0.141526	0.152339	0.814702	0.773481	0.900361	0.802415	0.241097	0.254545	0.181488
RidgeClassifier	0.135722	0.144850	0.804070	0.745856	0.893567	0.786654	0.232190	0.242588	0.172678
DecisionTreeClassifier	0.742525	0.675393	0.674596	0.712707	0.833140	0.849642	0.706168	0.693548	0.682540
RandomForestClassifier	0.946496	0.925000	0.563544	0.613260	0.975175	0.805656	0.705995	0.737542	0.839637
ExtraTreesClassifier	0.946006	0.953488	0.486842	0.453039	0.976802	0.726086	0.641973	0.614232	0.780952
AdaBoostClassifier	0.756803	0.734513	0.505333	0.458564	0.963798	0.726034	0.605044	0.564626	0.655608
KNeighborsClassifier	0.796448	0.718310	0.304175	0.281768	0.837694	0.638719	0.438324	0.404762	0.548387
SVC	0.277130	0.305243	0.915298	0.900552	0.964938	0.910116	0.425086	0.455944	0.351748

Conclusion: RandomForestClassifier selected

Highest f1-beta score. Low variance

04

Model Improvement

List of techniques used

- Pipeline
- Data augmentation
  - RandomOverSampler
- Feature Selection
  - Recursive Feature Elimination (RFE)
- Hyper-parameter tuning

Model Improvement Pipeline

Purpose:

-Simplify the data processing methods + prevent data leakage

```
cat_features = ['Quality']
num_features = ['Unique ID',"Ambient T (C)" ,"Rotation Speed (rpm)", "Torque (Nm)", "Tool Wear (min)",'Process T (C)','horsepower']


num_features_pipe = Pipeline([
    ('Itertive_impute',IterativeImputer()),
    ("scaler", StandardScaler())
])


cat_features_pipe = Pipeline([
    ('simple_impute',SimpleImputer(strategy='most_frequent')),
    ("onehot", OneHotEncoder(drop='first'))
])


prep_pipe = ColumnTransformer([
    ("num", num_features_pipe, num_features),
    ("cat", cat_features_pipe, cat_features)
])


pipe = Pipeline([
    ("preprocess", prep_pipe),
    ("estimator", RandomForestClassifier(random_state = 12,class_weight='balanced'))
])
```

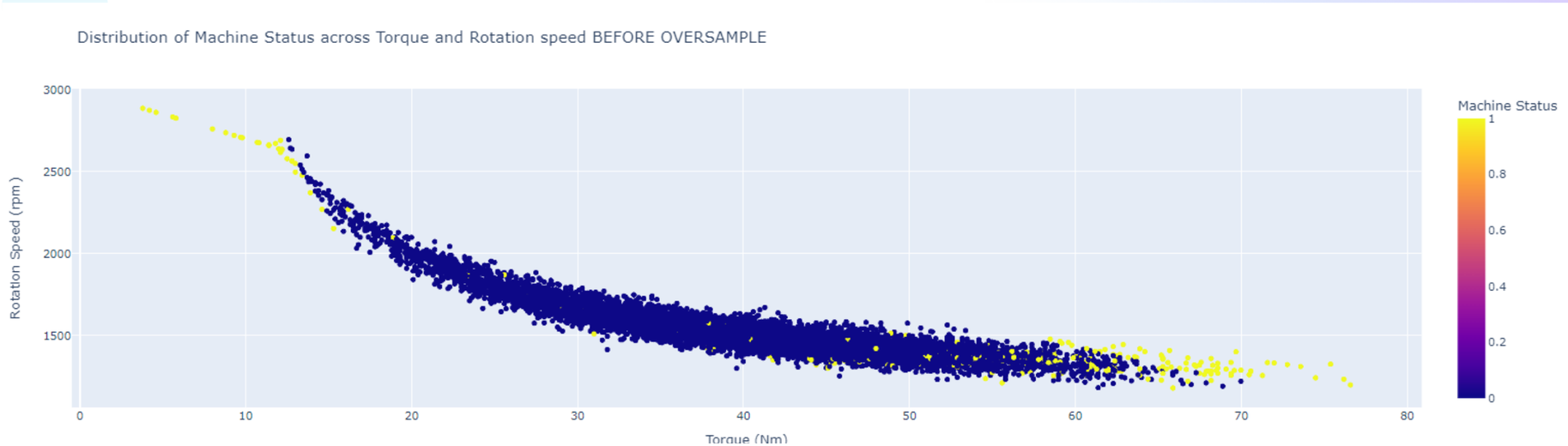
```
[214] pipe.fit(X_train_raw,y_train)
```

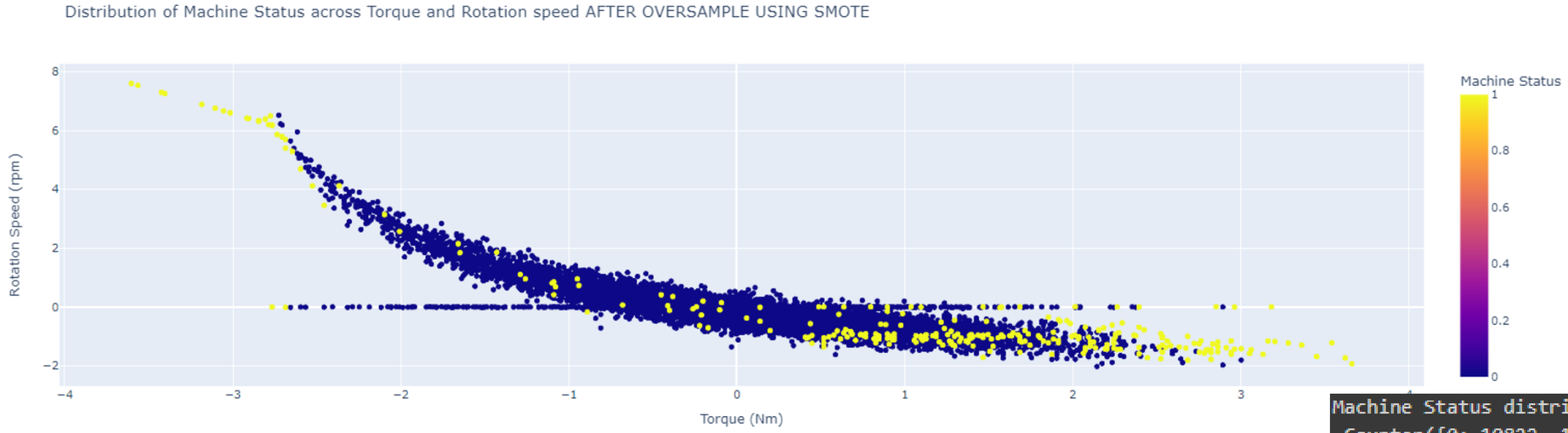
Model Improvement

Data augmentation

Note:

Since it's easiest to visualize a 2D plane, I just chose Torque and Rotation speed as both features have the highest correlation to Machine Status






```
Machine Status distribution BEFORE oversample:
 Counter({0: 10822, 1: 378})
Machine Status distribution AFTER oversample:
 Counter({0: 10822, 1: 10822})
```

Model Improvement

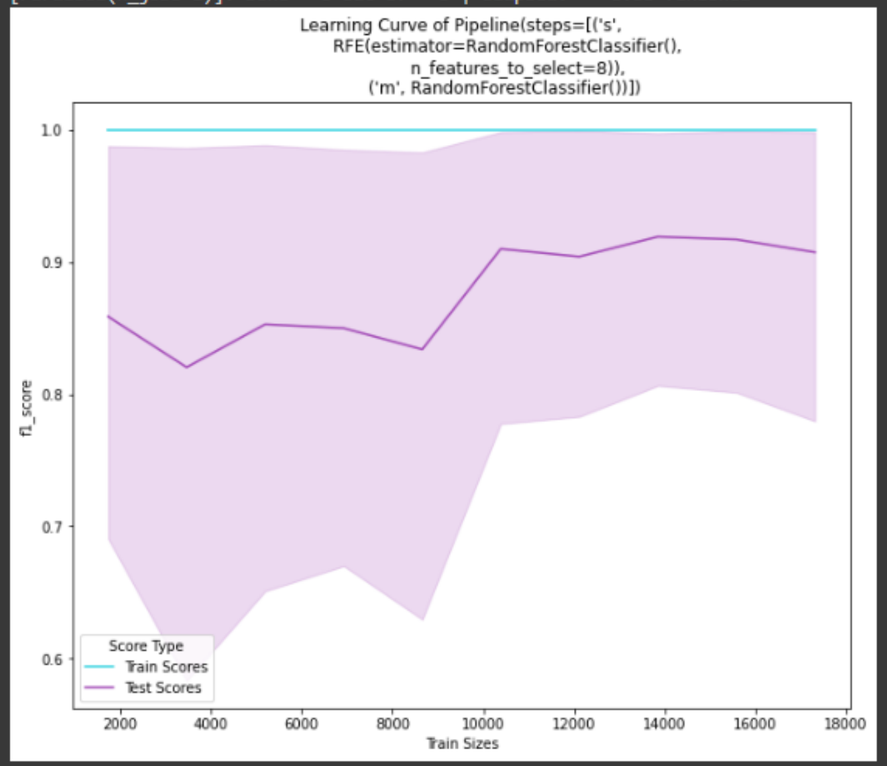
Data augmentation

CV test scores improved
However, there still seems to be a significant variance, as seen from the learning curve

≡

```
RF = RandomForestClassifier(class_weight='balanced')

single_evalutaion(RF, over_X_train, X_test, over_y_train, y_test,curve=True)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    8.0s finished
```



	auc_score	cv_auc	cv_f1_score	cv_precision	cv_recall	f1_beta_test_score	f1_test_score	prescision_test_score	recall_test_score
RandomForestClassifier	0.848989	1.0	0.998018	0.996044	1.0	0.838838	0.781538	0.881944	0.701657

Re-evaluted Randomforestclassifier

Model Improvement

Feature Selection

In order to select the optimal number of features to eliminate with RFE. Since we only have 8 features we can search every possible number of the RFE model to extract out.
We will select the number of features remaining that has the highest f1-score

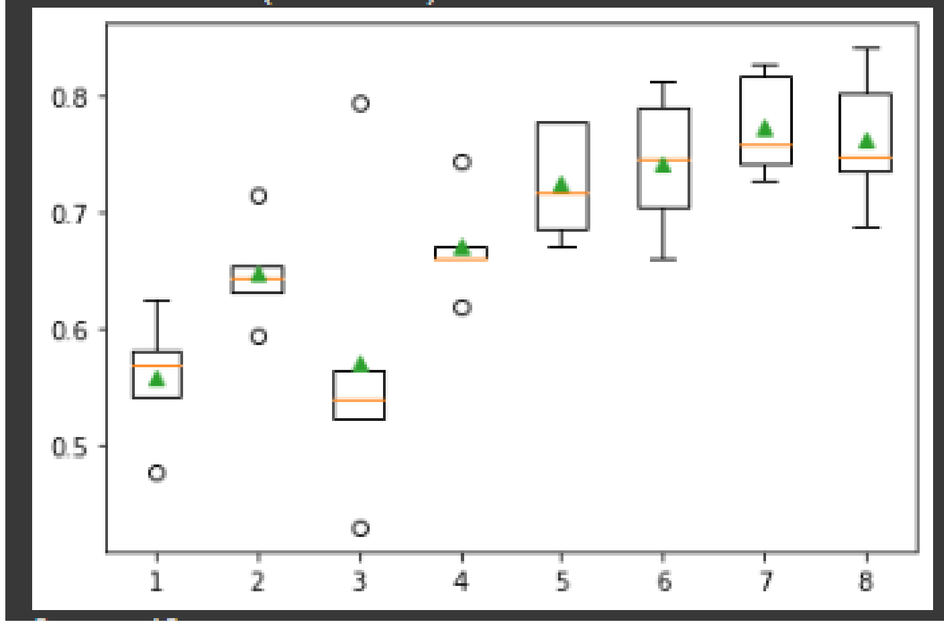
```
# explore the number of selected features for RFE
from numpy import mean
from numpy import std

from sklearn.feature_selection import RFE


# get a list of models to evaluate
def get_models():
  models = dict()
  for i in range(1, 9):
    rfe = RFE(estimator=RandomForestClassifier(), n_features_to_select=i)
    model = RandomForestClassifier()
    models[str(i)] = Pipeline(steps=[('s',rfe),('m',model)])
  return models

# evaluate a give model using cross-validation
def evaluate_model(model, X, y):
  cv = KFold(n_splits=5,shuffle=True,random_state=1)
  scores = cross_val_score(model, X, y, scoring='f1', cv=cv, n_jobs=-1, error_score='raise')
  return scores


# get the models to evaluate
models = get_models()
# evaluate the models and store results
results, names = list(), list()
for name, model in models.items():
  scores = evaluate_model(model, X_train, y_train)
  results.append(scores)
  names.append(name)
  print('>%s %.6f (%.6f)' % (name, mean(scores), std(scores)))

# plot model performance for comparison
pyplot.boxplot(results, labels=names, showmeans=True)
pyplot.show()
results
```

Model Improvement
Feature Selection

The mean f1 score stops improving at the 7th Feature. Furthermore, the s.d when all 8 features are kept is higher then only 7.

No. iteration to stop at = 7

```
>1 0.559632 (0.049341)
>2 0.648827 (0.039515)
>3 0.570711 (0.120652)
>4 0.671636 (0.040365)
>5 0.726337 (0.044583)
>6 0.743334 (0.055232)
>7 0.774743 (0.040089)
>8 0.763829 (0.053321)
```



Model Improvement
Hyper-parameter tuning

Method: **RandomSearchCV**
I chose RandomSearchCV instead of GridseachCV as it is more efficient through computing power and time.

We will try adjusting the following set of hyperparameters:

- n_estimators = number of trees in the forest
- max_features = max number of features considered for splitting a node
- max_depth = max number of levels in each decision tree
- min_samples_split = min number of data points placed in a node before the node is split
- min_samples_leaf = min number of data points allowed in a leaf node
- bootstrap = method for sampling data points (with or without replacement)

Model Improvement

Hyper-parameter tuning

```
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Class weight

# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
print(random_grid)
```

Model Improvement

Hyper-parameter tuning

```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 75, cv = 3, verbose=2, random_state=42, n_jobs = -1)
# Fit the random search model
rf_random.fit(over_X_train, over_y_train)

rf_random.best_params_
```

```
Fitting 3 folds for each of 75 candidates, totalling 225 fits
{'bootstrap': False,
 'max_depth': None,
 'max_features': 'sqrt',
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 400}
```

05

Final Model Evaluation

Conducted on Holdout test set

Finally, the Holdout test dataset can be used to act as a final sanity check to evaluate the model.

Compared against Baseline model

Chosen baseline: Dummy Classifier strategy='most_frequent'

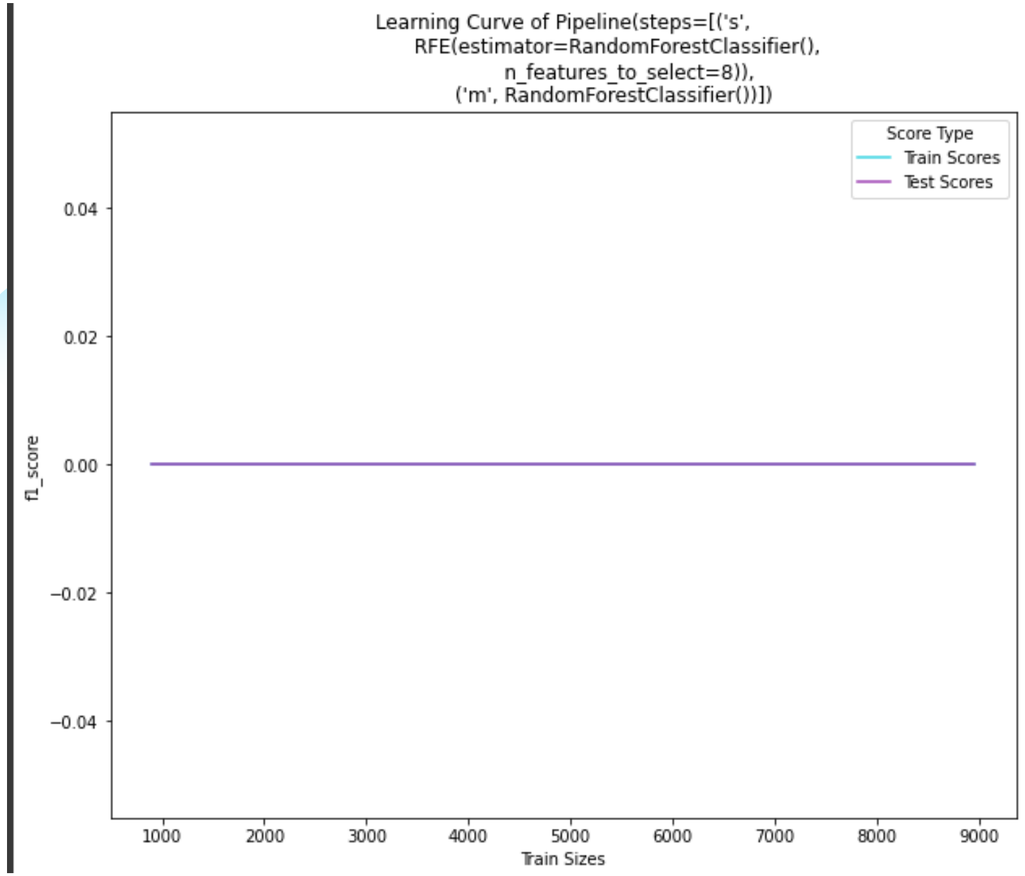
Plotted on confusion matrics

TO explain errors of finalzied model

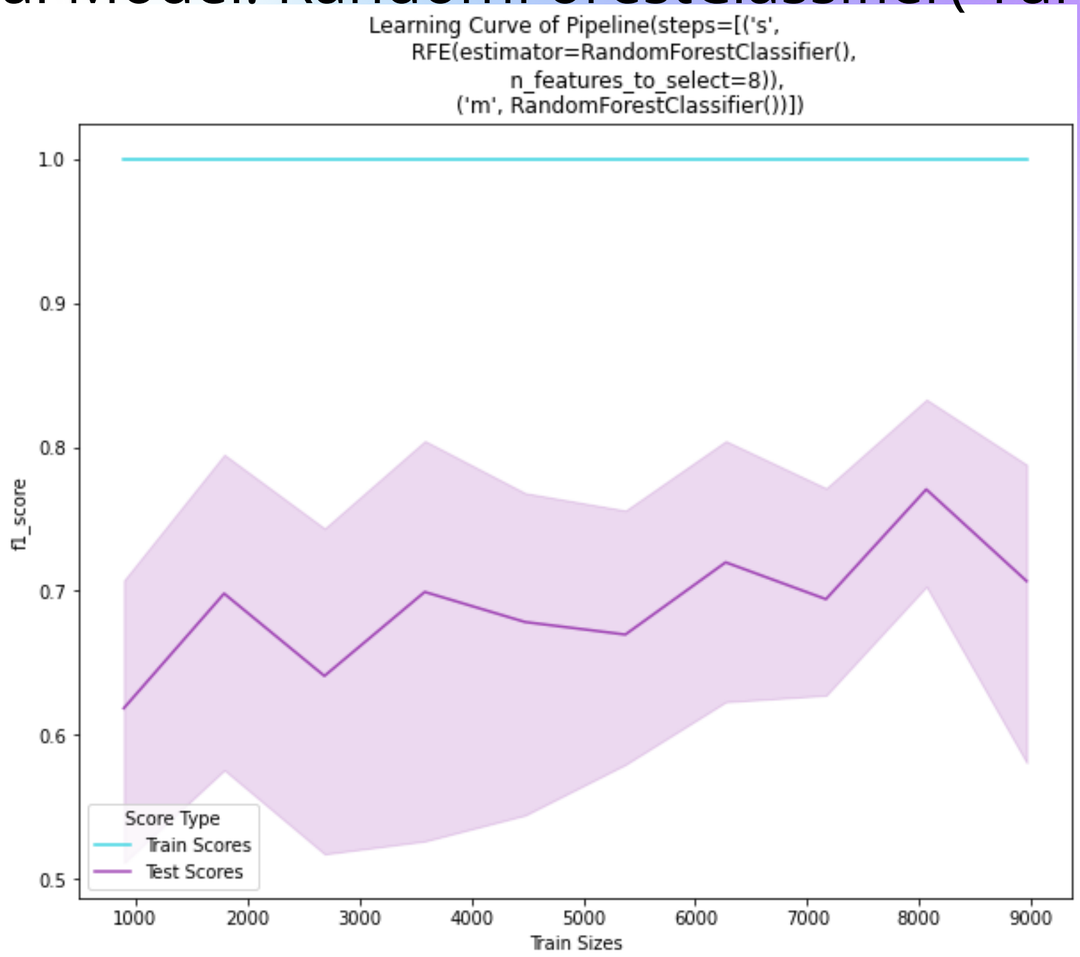
Compared against untuned model

Showcase use of hyperparameter tuning

Stupid baseline: DummyClassifier strategy = most_frequent



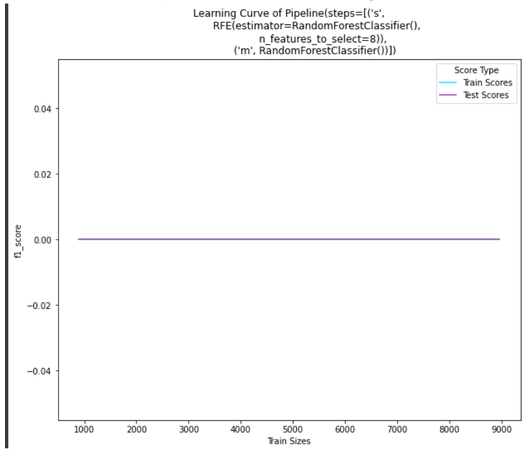
Final Model: RandomForestClassifier(Tuned)



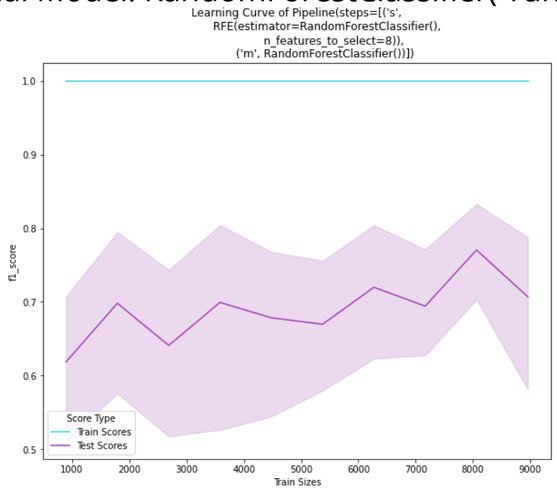
	auc_score	cv_auc	cv_f1_score	cv_precision	cv_recall	f1_beta_test_score	f1_test_score	prescision_test_score	recall_test_score
DummyClassifier(strategy='most_frequent')	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0

	auc_score	cv_auc	cv_f1_score	cv_precision	cv_recall	f1_beta_test_score	f1_test_score	prescision_test_score	recall_test_score
RandomForestClassifier(Tuned)	0.872495	0.975896	0.806577	0.927617	0.714316	0.915875	0.84375	0.971223	0.745856

Stupid baseline: DummyClassifier
strategy = most_frequent



Final Model: RandomForestClassifier(Tuned)



	auc_score	cv_auc	cv_f1_score	cv_precision	cv_recall	f1_beta_test_score	f1_test_score	prescision_test_score	recall_test_score
DummyClassifier(strategy='most_frequent')	0.5	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0

	auc_score	cv_auc	cv_f1_score	cv_precision	cv_recall	f1_beta_test_score	f1_test_score	prescision_test_score	recall_test_score
RandomForestClassifier(Tuned)	0.872495	0.975896	0.806577	0.927617	0.714316	0.915875	0.84375	0.971223	0.745856

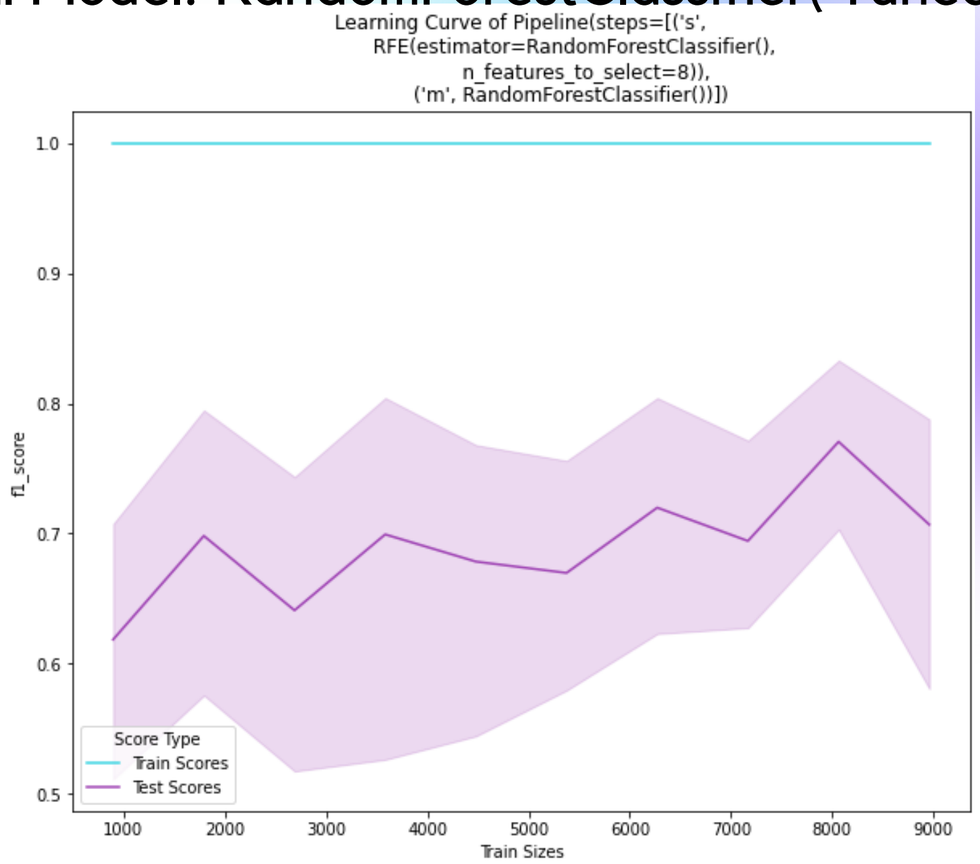
RandomforestClassifier has much smaller errors compared to DummyClassifier, based on fβ-score 0 compared to 0.91. However, the fitting time for RandomforestClassifier is high at 54s compared to 0.003 for baseline (which makes sense)

- For RandomforestClassifier , we can see a gap, relatively high variance, between training score and cross-validation score. Thus can conclude the model is slightly underfitted.

- For Dummy classifier, we can see a very high bias from the learning curve score

RandomForestClassifier (untuned)

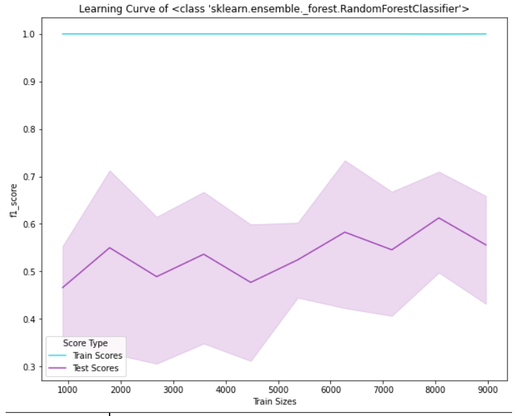
Final Model: RandomForestClassifier(Tuned)



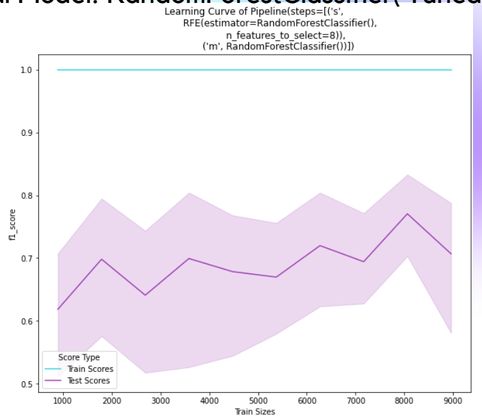
	cv_precision	prescision_test_score	cv_recall	recall_test_score	cv_auc	auc_score	cv_f1_score	f1_test_score	f1_beta_test_score
RandomForestClassifier	0.946496	0.925000	0.563544	0.613260	0.975175	0.805656	0.705995	0.737542	0.839637

	auc_score	cv_auc	cv_f1_score	cv_precision	cv_recall	f1_beta_test_score	f1_test_score	prescision_test_score	recall_test_score
RandomForestClassifier(Tuned)	0.872495	0.975896	0.806577	0.927617	0.714316	0.915875	0.84375	0.971223	0.745856

RandomForestClassifier (untuned)



Final Model: RandomForestClassifier(Tuned)



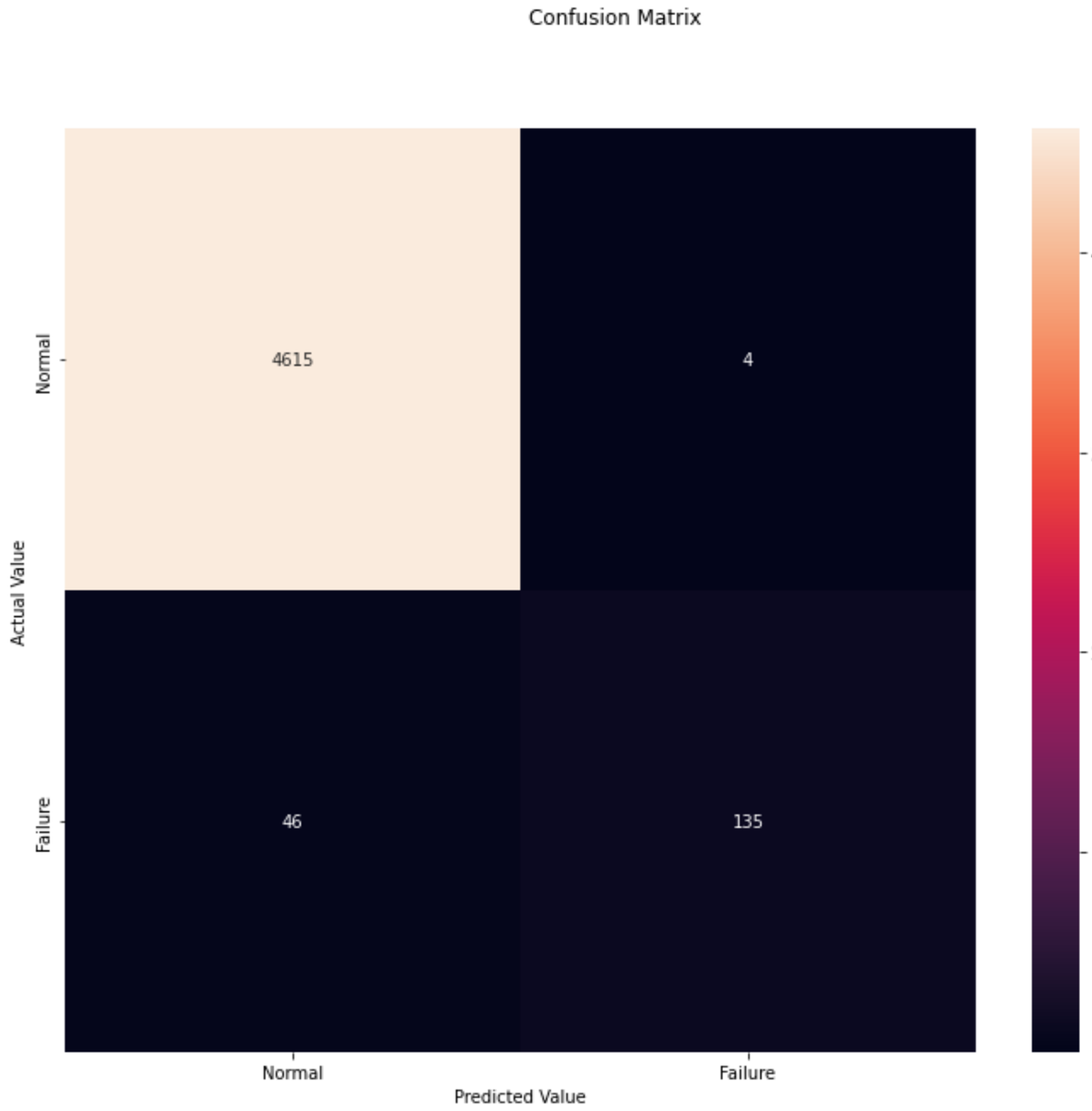
	auc_score	cv_auc	cv_f1_score	cv_precision	cv_recall	f1_beta_test_score	f1_test_score	prescision_test_score	recall_test_score
RandomForestClassifier(Tuned)	0.872495	0.975896	0.806577	0.927617	0.714316	0.915875	0.84375	0.971223	0.745856

	cv_precision	prescision_test_score	cv_recall	recall_test_score	cv_auc	auc_score	cv_f1_score	f1_test_score	f1_beta_test_score
RandomForestClassifier	0.946496	0.925000	0.563544	0.613260	0.975175	0.805656	0.705995	0.737542	0.839637

After evaluating the model with hold-out dev set, we noticed that RandomizedCV hyperparameters tuning have reduced the dev set error of a small margin as can be seen from the increase in fβ-score 0.839 --> 0.915

However, the variance still remains relatively high

Final Model Evaluation

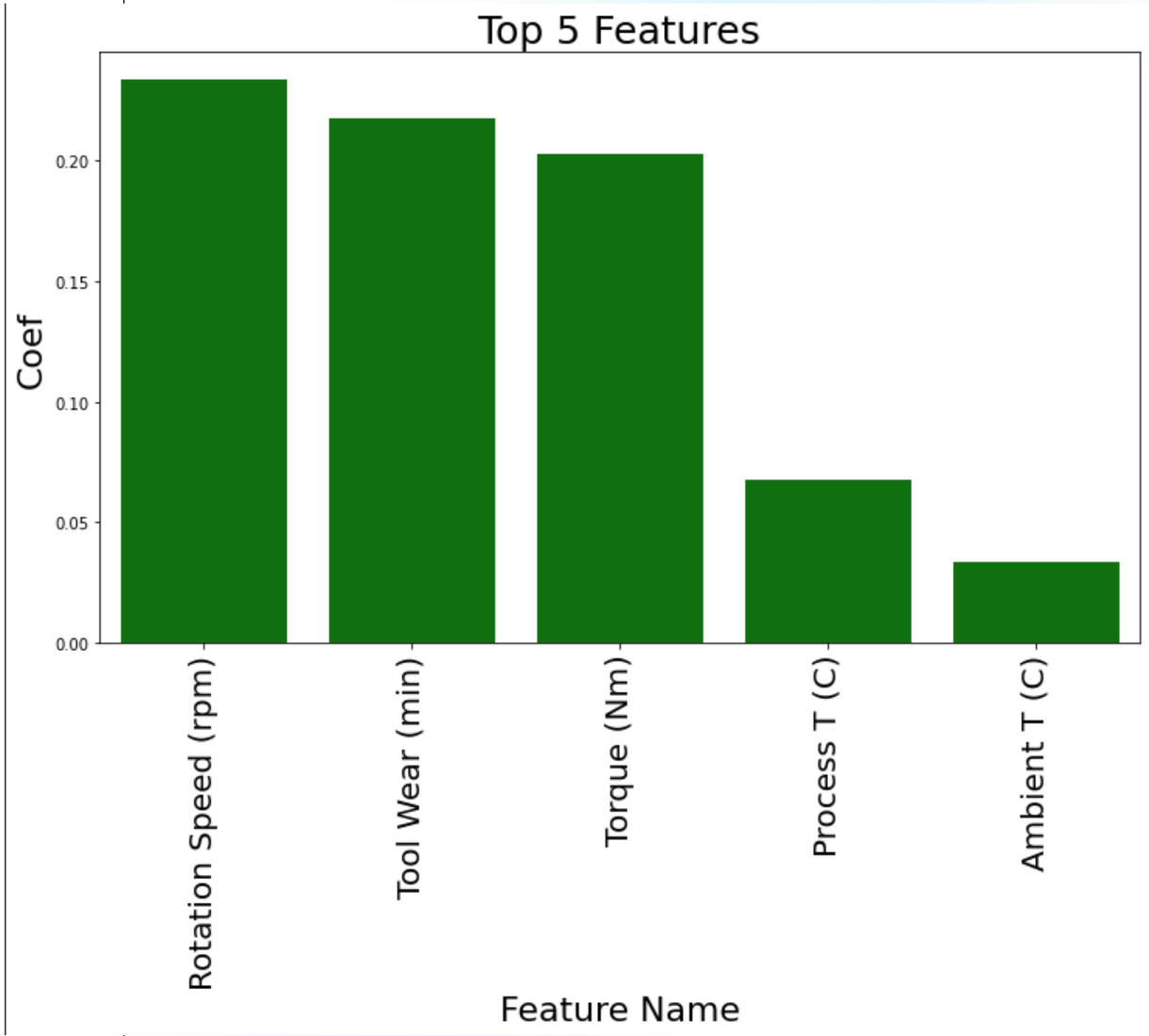


Generally we can see that our RandomforestClassifier is able to predict Normal functioning Machines to almost perfection. Only 4 predicted incorrectly out of 4.6k

However, the RandomForestClassifier performs poorly when classifying Machine Failure. around 25% of the time the model predicts a machine failure as Normal.

Since the Main Goal is to predict Machine Failure, I feel the model failed at its main prediction task. This could be due to the over simplicity of the model

Feature Importance



Our Final Model RandomForest Classifier ranks Rotation speed the most important feature at 0.25coef followed by Type-Notebook than Weight.

It's surprising that Torque is not the top important features given that it was highly correlated to Machine status in our EDA.

Learning Points

Dealing with Imbalanced classes

It was interesting to find solutions to handle imbalanced classes as I feel its a good skill to know in more useful cases like dealing with card fraud or Spam Email detection. Useful methods like selecting appropriateEvalution metrics, changing model param to "Weighted" and even oversampling methods can be used.