Binary Classification of Machine Failures

Import Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import random
import os
from copy import deepcopy
from functools import partial
from itertools import combinations
import random
import gc
import time
import math

# Import sklearn classes for model selection, cross validation, and performance evaluation
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold, KFold
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import StandardScaler, MinMaxScaler
import seaborn as sns
from category_encoders import OrdinalEncoder, CountEncoder, CatBoostEncoder, OneHotEncoder
from sklearn.preprocessing import FunctionTransformer, LabelEncoder # OneHotEncoder
from sklearn.compose import ColumnTransformer
from imblearn.under_sampling import RandomUnderSampler
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import SimpleImputer, KNNImputer, IterativeImputer
from sklearn.decomposition import PCA, NMF
from sklearn.pipeline import Pipeline
from sklearn.pipeline import make_pipeline
from sklearn.compose import make_column_transformer
from sklearn.compose import make_column_selector
import shap

# Import libraries for Hypertuning
import optuna

# Import libraries for gradient boosting
import lightgbm as lgb
import xgboost as xgb
from xgboost.callback import EarlyStopping
from sklearn.ensemble import RandomForestClassifier, HistGradientBoostingClassifier, GradientBoostingClassifier
from imblearn.ensemble import BalancedRandomForestClassifier
from sklearn.svm import NuSVC, SVC
from sklearn.linear_model import LogisticRegressionCV, LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from catboost import CatBoost, CatBoostRegressor, CatBoostClassifier
from catboost import Pool

from sklearn.feature_selection import RFE, RFECV
from sklearn.inspection import permutation_importance

# Suppress warnings
import warnings
warnings.filterwarnings("ignore", category=UserWarning)

from colorama import Style, Fore
blk = Style.BRIGHT + Fore.BLACK
red = Style.BRIGHT + Fore.RED
blu = Style.BRIGHT + Fore.BLUE
res = Style.RESET_ALL
```

```
/opt/conda/lib/python3.10/site-packages/shap/utils/_clustering.py:35: NumbaDeprecationWarning: The 'nopython' keyword a
rgument was not supplied to the 'numba.jit' decorator. The implicit default value for this argument is currently False,
but it will be changed to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/deprecation.html#d
eprecation-of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _pt_shuffle_rec(i, indexes, index_mask, partition_tree, M, pos):
/opt/conda/lib/python3.10/site-packages/shap/utils/_clustering.py:54: NumbaDeprecationWarning: The 'nopython' keyword a
rgument was not supplied to the 'numba.jit' decorator. The implicit default value for this argument is currently False,
but it will be changed to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/deprecation.html#d
eprecation-of-object-mode-fall-back-behaviour-when-using-jit for details.
  def delta_minimization_order(all_masks, max_swap_size=100, num_passes=2):
/opt/conda/lib/python3.10/site-packages/shap/utils/_clustering.py:63: NumbaDeprecationWarning: The 'nopython' keyword a
rgument was not supplied to the 'numba.jit' decorator. The implicit default value for this argument is currently False,
but it will be changed to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/deprecation.html#d
eprecation-of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _reverse_window(order, start, length):
/opt/conda/lib/python3.10/site-packages/shap/utils/_clustering.py:69: NumbaDeprecationWarning: The 'nopython' keyword a
rgument was not supplied to the 'numba.jit' decorator. The implicit default value for this argument is currently False,
but it will be changed to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/deprecation.html#d
eprecation-of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _reverse_window_score_gain(masks, order, start, length):
/opt/conda/lib/python3.10/site-packages/shap/utils/_clustering.py:77: NumbaDeprecationWarning: The 'nopython' keyword a
rgument was not supplied to the 'numba.jit' decorator. The implicit default value for this argument is currently False,
but it will be changed to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/deprecation.html#d
eprecation-of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _mask_delta_score(m1, m2):
/opt/conda/lib/python3.10/site-packages/shap/links.py:5: NumbaDeprecationWarning: The 'nopython' keyword argument was n
ot supplied to the 'numba.jit' decorator. The implicit default value for this argument is currently False, but it will
be changed to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-o
f-object-mode-fall-back-behaviour-when-using-jit for details.
  def identity(x):
/opt/conda/lib/python3.10/site-packages/shap/links.py:10: NumbaDeprecationWarning: The 'nopython' keyword argument was
not supplied to the 'numba.jit' decorator. The implicit default value for this argument is currently False, but it will
be changed to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-o
f-object-mode-fall-back-behaviour-when-using-jit for details.
  def _identity_inverse(x):
/opt/conda/lib/python3.10/site-packages/shap/links.py:15: NumbaDeprecationWarning: The 'nopython' keyword argument was
not supplied to the 'numba.jit' decorator. The implicit default value for this argument is currently False, but it will
be changed to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-o
f-object-mode-fall-back-behaviour-when-using-jit for details.
  def logit(x):
/opt/conda/lib/python3.10/site-packages/shap/links.py:20: NumbaDeprecationWarning: The 'nopython' keyword argument was
not supplied to the 'numba.jit' decorator. The implicit default value for this argument is currently False, but it will
be changed to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/deprecation.html#deprecation-o
f-object-mode-fall-back-behaviour-when-using-jit for details.
  def _logit_inverse(x):
/opt/conda/lib/python3.10/site-packages/shap/utils/_masked_model.py:363: NumbaDeprecationWarning: The 'nopython' keywor
d argument was not supplied to the 'numba.jit' decorator. The implicit default value for this argument is currently Fal
se, but it will be changed to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/deprecation.ht
ml#deprecation-of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _build_fixed_single_output(averaged_outs, last_outs, outputs, batch_positions, varying_rows, num_varying_rows, li
nk, linearizing_weights):
/opt/conda/lib/python3.10/site-packages/shap/utils/_masked_model.py:385: NumbaDeprecationWarning: The 'nopython' keywor
d argument was not supplied to the 'numba.jit' decorator. The implicit default value for this argument is currently Fal
se, but it will be changed to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/deprecation.ht
ml#deprecation-of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _build_fixed_multi_output(averaged_outs, last_outs, outputs, batch_positions, varying_rows, num_varying_rows, lin
k, linearizing_weights):
/opt/conda/lib/python3.10/site-packages/shap/utils/_masked_model.py:428: NumbaDeprecationWarning: The 'nopython' keywor
d argument was not supplied to the 'numba.jit' decorator. The implicit default value for this argument is currently Fal
se, but it will be changed to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/deprecation.ht
ml#deprecation-of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _init_masks(cluster_matrix, M, indices_row_pos, indptr):
/opt/conda/lib/python3.10/site-packages/shap/utils/_masked_model.py:439: NumbaDeprecationWarning: The 'nopython' keywor
d argument was not supplied to the 'numba.jit' decorator. The implicit default value for this argument is currently Fal
se, but it will be changed to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/deprecation.ht
ml#deprecation-of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _rec_fill_masks(cluster_matrix, indices_row_pos, indptr, indices, M, ind):
/opt/conda/lib/python3.10/site-packages/shap/maskers/_tabular.py:186: NumbaDeprecationWarning: The 'nopython' keyword a
rgument was not supplied to the 'numba.jit' decorator. The implicit default value for this argument is currently False,
but it will be changed to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/deprecation.html#d
eprecation-of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _single_delta_mask(dind, masked_inputs, last_mask, data, x, noop_code):
/opt/conda/lib/python3.10/site-packages/shap/maskers/_tabular.py:197: NumbaDeprecationWarning: The 'nopython' keyword a
rgument was not supplied to the 'numba.jit' decorator. The implicit default value for this argument is currently False,
but it will be changed to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/deprecation.html#d
eprecation-of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _delta_masking(masks, x, curr_delta_inds, varying_rows_out,
/opt/conda/lib/python3.10/site-packages/shap/maskers/_image.py:175: NumbaDeprecationWarning: The 'nopython' keyword arg
ument was not supplied to the 'numba.jit' decorator. The implicit default value for this argument is currently False, b
ut it will be changed to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/deprecation.html#de
precation-of-object-mode-fall-back-behaviour-when-using-jit for details.
  def _jit_build_partition_tree(xmin, xmax, ymin, ymax, zmin, zmax, total_ywidth, total_zwidth, M, clustering, q):
/opt/conda/lib/python3.10/site-packages/shap/explainers/_partition.py:676: NumbaDeprecationWarning: The 'nopython' keyw
ord argument was not supplied to the 'numba.jit' decorator. The implicit default value for this argument is currently F
alse, but it will be changed to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/reference/deprecation.
html#deprecation-of-object-mode-fall-back-behaviour-when-using-jit for details.
  def lower_credit(i, value, M, values, clustering):
The 'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The implicit default value for this argu
ment is currently False, but it will be changed to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/ref
erence/deprecation.html#deprecation-of-object-mode-fall-back-behaviour-when-using-jit for details.
The 'nopython' keyword argument was not supplied to the 'numba.jit' decorator. The implicit default value for this argu
ment is currently False, but it will be changed to True in Numba 0.59.0. See https://numba.readthedocs.io/en/stable/ref
erence/deprecation.html#deprecation-of-object-mode-fall-back-behaviour-when-using-jit for details.
```

Read data

In [2]:
```
filepath = '/input/playground-series-s3e17'

df_train = pd.read_csv(os.path.join(filepath, 'train.csv'), index_col=[0])
df_test = pd.read_csv(os.path.join(filepath, 'test.csv'), index_col=[0])
original = pd.read_csv('/input/machine-failure-predictions/machine failure.csv', index_col=[0])

target_col = 'Machine failure'
# num_cols = df_test.select_dtypes(include=['int64']).columns.tolist()
num_cols = [
    'Air temperature [K]',
    'Process temperature [K]',
    'Rotational speed [rpm]',
    'Torque [Nm]',
    'Tool wear [min]'
]
binary_cols = [
    'TWF',
    'HDF',
    'PWF',
    'OSF',
    'RNF'
]
cat_cols = df_test.select_dtypes(include=['object']).columns.tolist()

df_train['is_generated'] = 1
df_test['is_generated'] = 1
original['is_generated'] = 0

print(f"train shape :{df_train.shape}, ", f"test shape :{df_test.shape}")
print(f"original shape :{original.shape}")
```

```
train shape :(136429, 14),  test shape :(90954, 13)
original shape :(10000, 14)
```

In [3]:
```
def set_frame_style(df, caption=""):
    """Helper function to set dataframe presentation style.
    """
    return df.style.background_gradient(cmap='Blues').set_caption(caption).set_table_styles([{
    'selector': 'caption',
    'props': [
        ('color', 'Blue'),
        ('font-size', '18px'),
        ('font-weight','bold')
    ]}])

def check_data(data, title):
    cols = data.columns.to_list()
    display(set_frame_style(data[cols].head(),f'{title}: First 5 Rows Of Data'))
    display(set_frame_style(data[cols].describe(),f'{title}: Summary Statistics'))
    display(set_frame_style(data[cols].nunique().to_frame().rename({0:'Unique Value Count'}, axis=1).transpose(), f'{ti
    display(set_frame_style(data[cols].isna().sum().to_frame().transpose(), f'{title}:Columns With Nan'))

check_data(df_train, 'Train data')
print('-'*100)
check_data(df_test, 'Test data')
print('-'*100)
check_data(original, 'Original data')
```

Train data: First 5 Rows Of Data

id	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	TWF	HDF	PWF	OSF	RNF	is_generated
0	L50096	L	300.600000	309.600000	1596	36.100000	140	0	0	0	0	0	0	1
1	M20343	M	302.600000	312.100000	1759	29.100000	200	0	0	0	0	0	0	1
2	L49454	L	299.300000	308.500000	1805	26.500000	25	0	0	0	0	0	0	1
3	L53355	L	301.000000	310.900000	1524	44.300000	197	0	0	0	0	0	0	1
4	M24050	M	298.000000	309.000000	1641	35.400000	34	0	0	0	0	0	0	1

Train data: Summary Statistics

	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	TWF	HDF	PW
count	136429.000000	136429.000000	136429.000000	136429.000000	136429.000000	136429.000000	136429.000000	136429.000000	136429.00000
mean	299.862776	309.941070	1520.331110	40.348643	104.408901	0.015744	0.001554	0.005160	0.00239
std	1.862247	1.385173	138.736632	8.502229	63.965040	0.124486	0.039389	0.071649	0.04889
min	295.300000	305.800000	1181.000000	3.800000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	298.300000	308.700000	1432.000000	34.600000	48.000000	0.000000	0.000000	0.000000	0.00000
50%	300.000000	310.000000	1493.000000	40.400000	106.000000	0.000000	0.000000	0.000000	0.00000
75%	301.200000	310.900000	1580.000000	46.100000	159.000000	0.000000	0.000000	0.000000	0.00000
max	304.400000	313.800000	2886.000000	76.600000	253.000000	1.000000	1.000000	1.000000	1.00000

Train data: Unique Value Counts In Each Column

	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	TWF	HDF	PWF	OSF	RNF	is_generated
Unique Value Count	9976	3	95	81	952	611	246	2	2	2	2	2	2	1

Train data:Columns With Nan

	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	TWF	HDF	PWF	OSF	RNF	is_generated
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Test data: First 5 Rows Of Data

id	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	TWF	HDF	PWF	OSF	RNF	is_generated
136429	L50896	L	302.300000	311.500000	1499	38.000000	60	0	0	0	0	0	1
136430	L53866	L	301.700000	311.000000	1713	28.800000	17	0	0	0	0	0	1
136431	L50498	L	301.300000	310.400000	1525	37.700000	96	0	0	0	0	0	1
136432	M21232	M	300.100000	309.600000	1479	47.600000	5	0	0	0	0	0	1
136433	M19751	M	303.400000	312.300000	1515	41.300000	114	0	0	0	0	0	1

Test data: Summary Statistics

	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	TWF	HDF	PWF	OSF	
count	90954.000000	90954.000000	90954.000000	90954.000000	90954.000000	90954.000000	90954.000000	90954.000000	90954.000000	90954.
mean	299.859493	309.939375	1520.528179	40.335191	104.293962	0.001473	0.005343	0.002353	0.003870	0.(
std	1.857562	1.385296	139.970419	8.504683	63.871092	0.038355	0.072903	0.048449	0.062090	0.(
min	295.300000	305.700000	1168.000000	3.800000	0.000000	0.000000	0.000000	0.000000	0.000000	0.(
25%	298.300000	308.700000	1432.000000	34.600000	48.000000	0.000000	0.000000	0.000000	0.000000	0.(
50%	300.000000	310.000000	1493.000000	40.500000	106.000000	0.000000	0.000000	0.000000	0.000000	0.(
75%	301.200000	310.900000	1579.000000	46.200000	158.000000	0.000000	0.000000	0.000000	0.000000	0.(
max	304.400000	313.800000	2886.000000	76.600000	253.000000	1.000000	1.000000	1.000000	1.000000	1.(

Test data: Unique Value Counts In Each Column

	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	TWF	HDF	PWF	OSF	RNF	is_generated
Unique Value Count	9909	3	92	84	946	595	246	2	2	2	2	2	1

Test data:Columns With Nan

	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	TWF	HDF	PWF	OSF	RNF	is_generated
0	0	0	0	0	0	0	0	0	0	0	0	0	0

Original data: First 5 Rows Of Data

UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	TWF	HDF	PWF	OSF	RNF	is_generated
1	M14860	M	298.100000	308.600000	1551	42.800000	0	0	0	0	0	0	0	0
2	L47181	L	298.200000	308.700000	1408	46.300000	3	0	0	0	0	0	0	0
3	L47182	L	298.100000	308.500000	1498	49.400000	5	0	0	0	0	0	0	0
4	L47183	L	298.200000	308.600000	1433	39.500000	7	0	0	0	0	0	0	0
5	L47184	L	298.200000	308.700000	1408	40.000000	9	0	0	0	0	0	0	0

Original data: Summary Statistics

	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	TWF	HDF	PWF	
count	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.0
mean	300.004930	310.005560	1538.776100	39.986910	107.951000	0.033900	0.004600	0.011500	0.009500	0.0
std	2.000259	1.483734	179.284096	9.968934	63.654147	0.180981	0.067671	0.106625	0.097009	0.0
min	295.300000	305.700000	1168.000000	3.800000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	298.300000	308.800000	1423.000000	33.200000	53.000000	0.000000	0.000000	0.000000	0.000000	0.0
50%	300.100000	310.100000	1503.000000	40.100000	108.000000	0.000000	0.000000	0.000000	0.000000	0.0
75%	301.500000	311.100000	1612.000000	46.800000	162.000000	0.000000	0.000000	0.000000	0.000000	0.0
max	304.500000	313.800000	2886.000000	76.600000	253.000000	1.000000	1.000000	1.000000	1.000000	1.0

Original data: Unique Value Counts In Each Column

	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	TWF	HDF	PWF	OSF	RNF	is_generated
Unique Value Count	10000	3	93	82	941	577	246	2	2	2	2	2	2	1

Original data:Columns With Nan

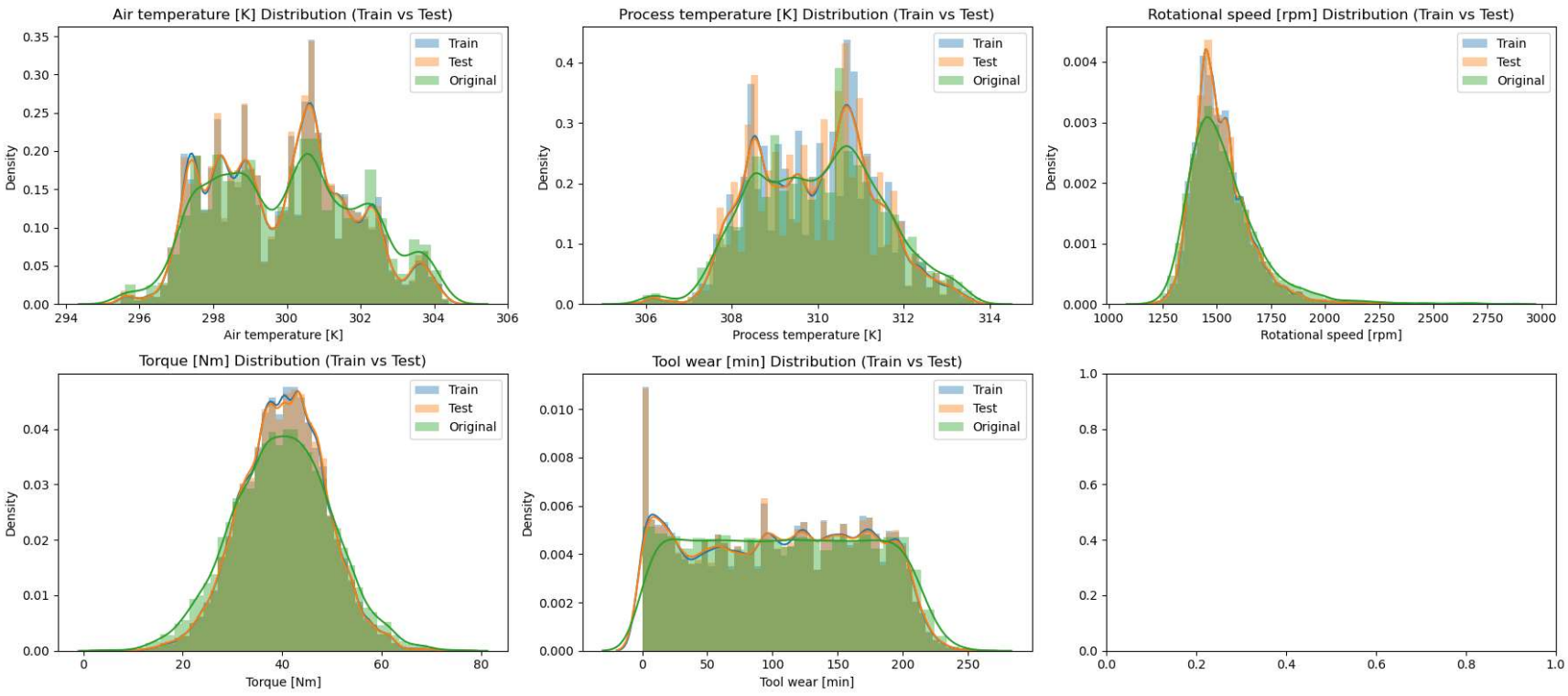
	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	TWF	HDF	PWF	OSF	RNF	is_generated
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

EDA

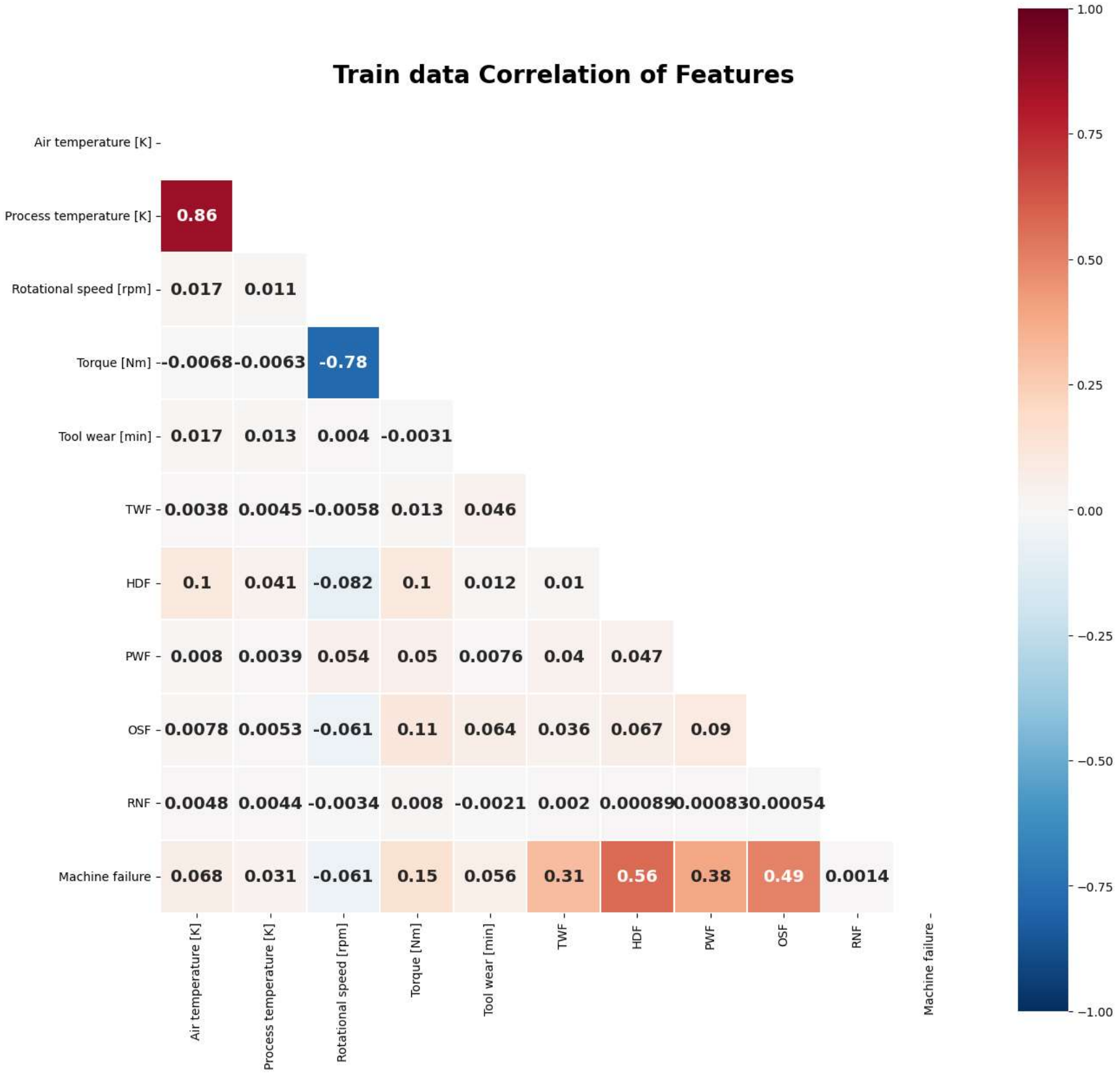
Contents:

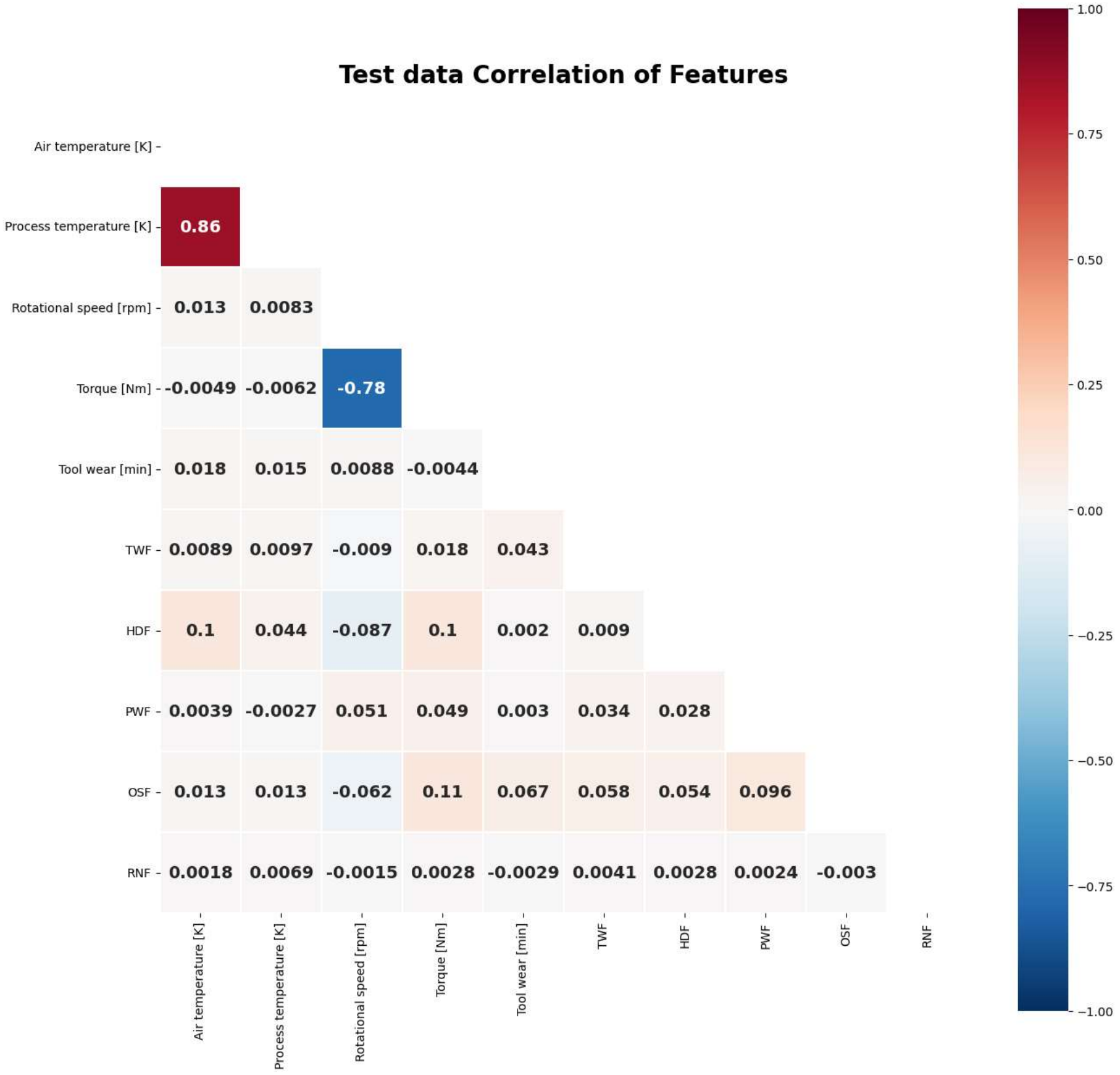
1. Train, Test and Original data histograms
2. Correlation of Features
3. Scatter plots of features by Machine failure (random undersampling)
4. Hierarchical Clustering
5. Pie and bar charts for categorical column features
6. Distribution Plot by Type
7. Boxplot by Machine failure
8. Violinplot by Machine failure
9. Scatter plots after dimensionality reduction with PCA by Machine failure

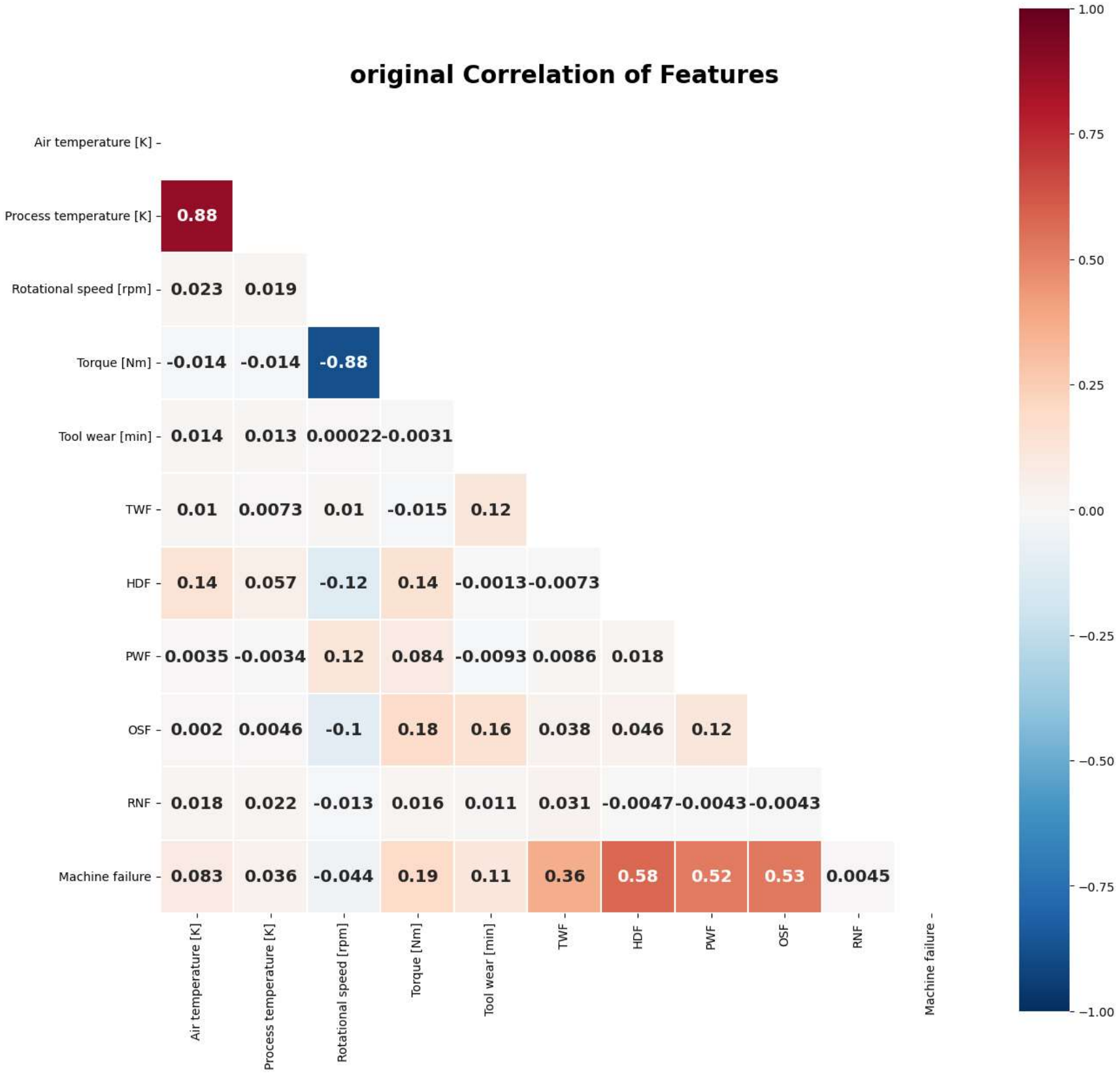
In [4]:
```
def plot_histograms(df_train, df_test, original, target_col, n_cols=3):
    n_rows = (len(df_train.columns) - 1) // n_cols + 1

    fig, axes = plt.subplots(nrows=n_rows, ncols=n_cols, figsize=(18, 4*n_rows))
    axes = axes.flatten()

    for i, var_name in enumerate(df_train.columns.tolist()):
        if var_name != 'is_generated':
            ax = axes[i]
            sns.distplot(df_train[var_name], kde=True, ax=ax, label='Train')
            if var_name != target_col:
                sns.distplot(df_test[var_name], kde=True, ax=ax, label='Test')
            sns.distplot(original[var_name], kde=True, ax=ax, label='Original')
            ax.set_title(f'{var_name} Distribution (Train vs Test)')
            ax.legend()

    plt.tight_layout()
    plt.show()

plot_histograms(df_train[num_cols], df_test[num_cols], original[num_cols], target_col, n_cols=3)
```



In [5]:
```python
def plot_heatmap(df, title):
    # Create a mask for the diagonal elements
    mask = np.zeros_like(df.astype(float).corr())
    mask[np.triu_indices_from(mask)] = True

    # Set the colormap and figure size
    colormap = plt.cm.RdBu_r
    plt.figure(figsize=(15, 15))

    # Set the title and font properties
    plt.title(f'{title} Correlation of Features', fontweight='bold', y=1.02, size=20)

    # Plot the heatmap with the masked diagonal elements
    sns.heatmap(df.astype(float).corr(), linewidths=0.1, vmax=1.0, vmin=-1.0,
                square=True, cmap=colormap, linecolor='white', annot=True, annot_kws={"size": 14, "weight": "bold"},
                mask=mask)

plot_heatmap(df_train[num_cols+binary_cols+[target_col]], title='Train data')
plot_heatmap(df_test[num_cols+binary_cols], title='Test data')
plot_heatmap(original[num_cols+binary_cols+[target_col]], title='original')
```

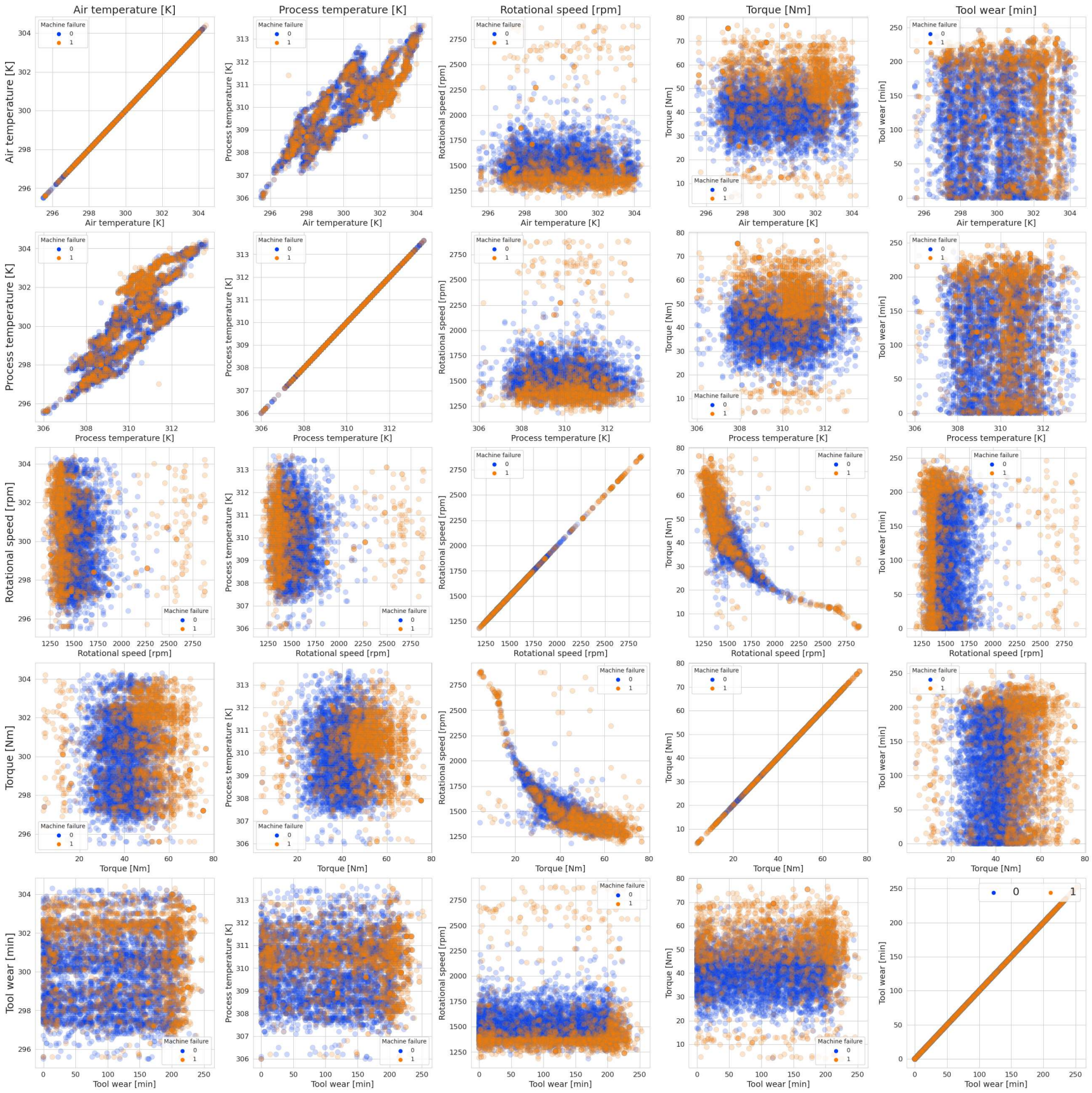

Train data Correlation of Features
Air temperature [K]
Process temperature [K]
Rotational speed [rpm]
Torque [Nm]
Tool wear [min]
TWF
HDF
PWF
OSF
RNF
Machine failure
0.86
0.017 0.011
-0.0068 -0.0063 -0.78
0.017 0.013 0.004 -0.0031
0.0038 0.0045 -0.0058 0.013 0.046
0.1 0.041 -0.082 0.1 0.012 0.01
0.008 0.0039 0.054 0.05 0.0076 0.04 0.047
0.0078 0.0053 -0.061 0.11 0.064 0.036 0.067 0.09
0.0048 0.0044 -0.0034 0.008 -0.0021 0.002 0.00089 0.00083 0.00054
0.068 0.031 -0.061 0.15 0.056 0.31 0.56 0.38 0.49 0.0014
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00


Test data Correlation of Features
Air temperature [K]
Process temperature [K]
Rotational speed [rpm]
Torque [Nm]
Tool wear [min]
TWF
HDF
PWF
OSF
RNF
0.86
0.013 0.0083
-0.0049 -0.0062 -0.78
0.018 0.015 0.0088 -0.0044
0.0089 0.0097 -0.009 0.018 0.043
0.1 0.044 -0.087 0.1 0.002 0.009
0.0039 -0.0027 0.051 0.049 0.003 0.034 0.028
0.013 0.013 -0.062 0.11 0.067 0.058 0.054 0.096
0.0018 0.0069 -0.0015 0.0028 -0.0029 0.0041 0.0028 0.0024 -0.003
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00



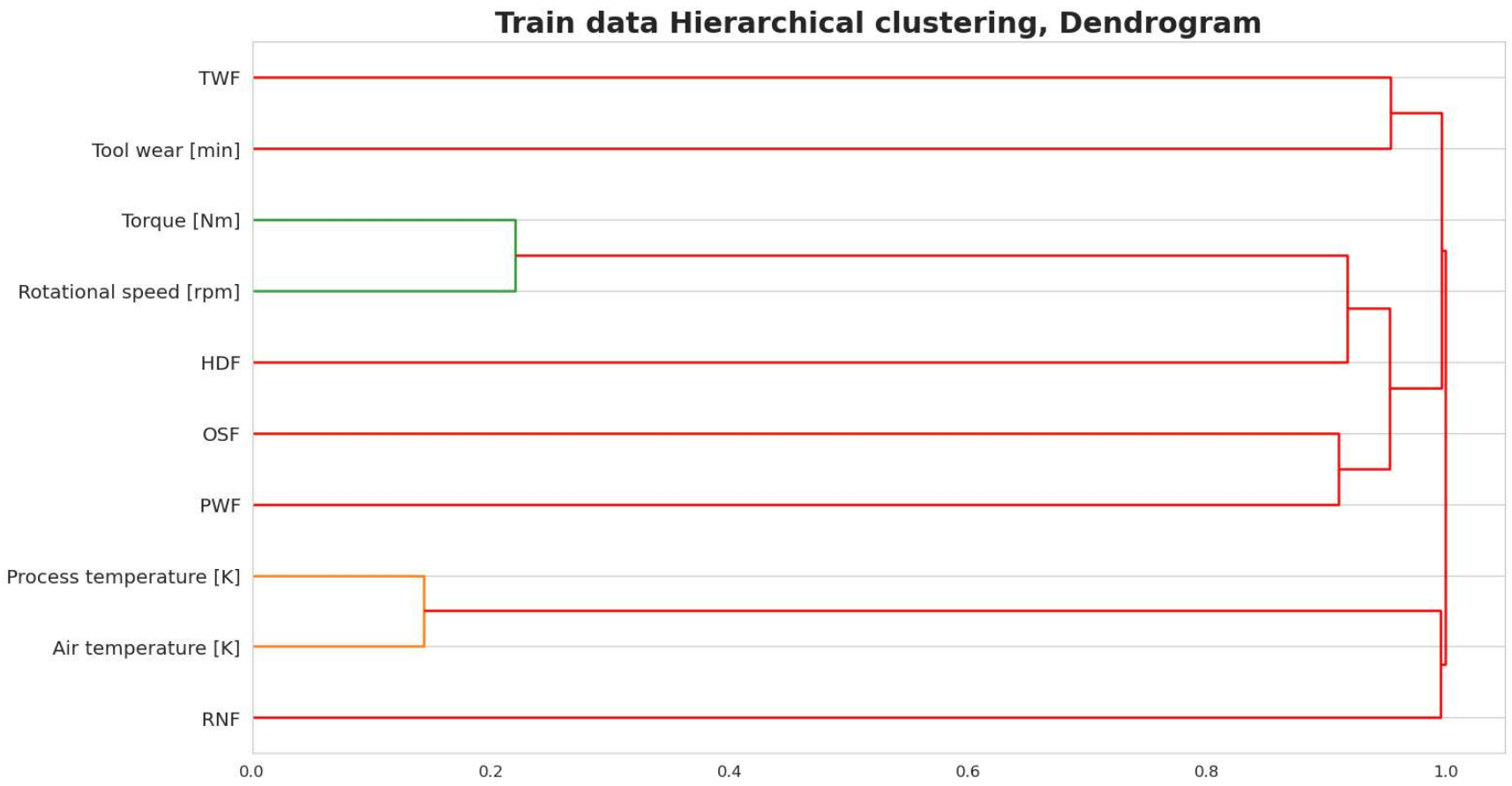
In [6]:

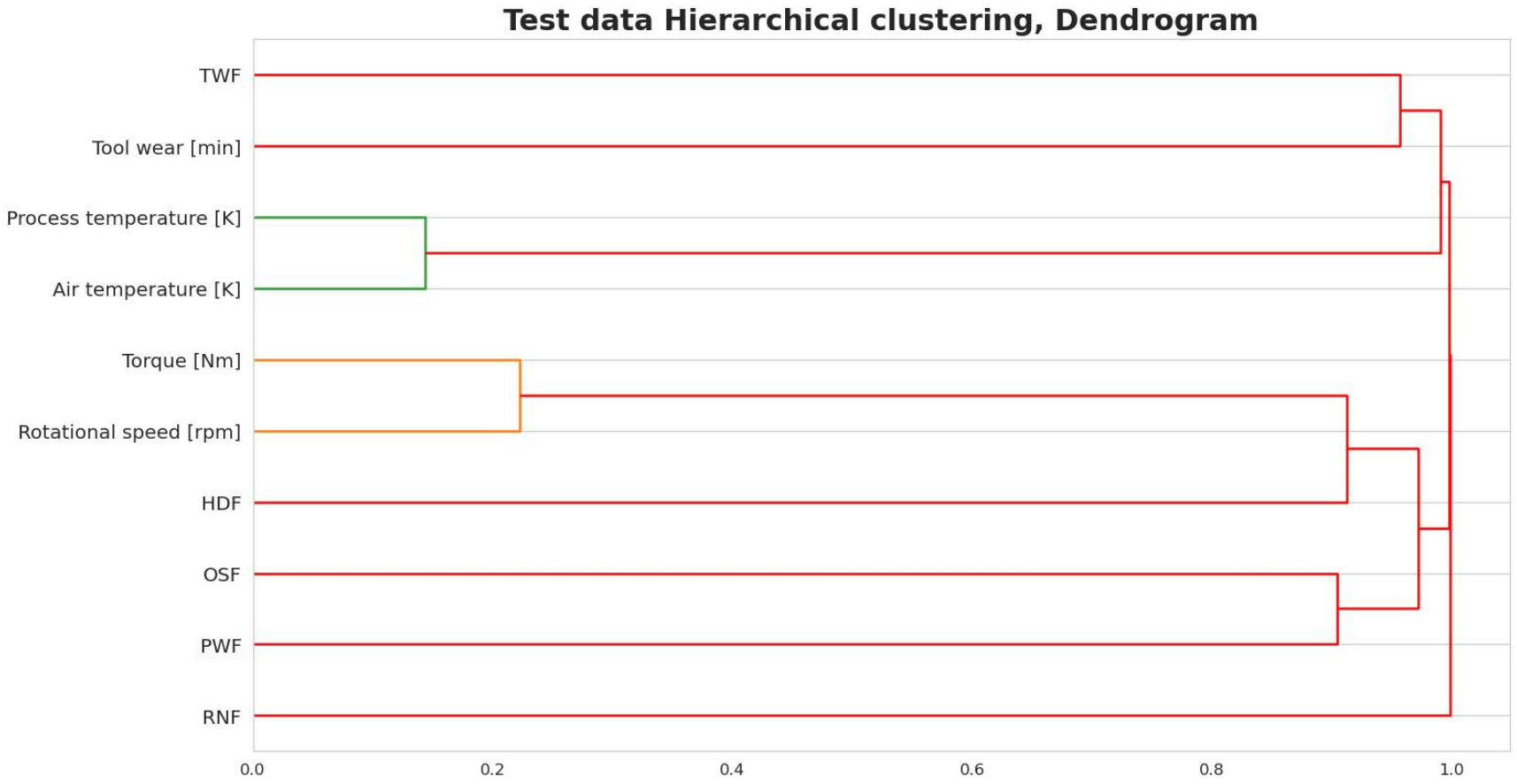
```python
def plot_scatter_matrix(df, target_col, drop_cols=[], size=26):
    # sns.pairplot()

    sns.set_style('whitegrid')
    cols = df.columns.drop([target_col] + drop_cols)
    fig, axes = plt.subplots(len(cols), len(cols), figsize=(size, size), sharex=False, sharey=False)

    for i, col in enumerate(cols):
        for j, col_ in enumerate(cols):
            axes[i,j].set_xlabel(f'{col}', fontsize=14)
            axes[i,j].set_ylabel(f'{col_}', fontsize=14)

            # Plot the scatterplot
            sns.scatterplot(data=df, x=col, y=col_, hue=target_col, ax=axes[i,j],
                            s=80, edgecolor='gray', alpha=0.2, palette='bright')

            axes[i,j].tick_params(axis='both', which='major', labelsize=12)

            if i == 0:
                axes[i,j].set_title(f'{col_}', fontsize=18)
            if j == 0:
                axes[i,j].set_ylabel(f'{col}', fontsize=18)

    plt.tight_layout(pad=0.5, h_pad=0.5, w_pad=0.5)
    plt.legend(loc='upper right', ncol=5, fontsize=18)
    plt.show()

sampling_strategy = 0.5
rus = RandomUnderSampler(sampling_strategy=sampling_strategy, random_state=42)
X_train_res, y_train_res = rus.fit_resample(df_train.drop(target_col, axis=1), df_train[target_col])
plot_scatter_matrix(pd.concat([X_train_res[num_cols], y_train_res], axis=1), target_col)

# del X_train_res, y_train_res, rus
```



In [7]:

```
from scipy.cluster import hierarchy
from scipy.cluster.hierarchy import dendrogram, linkage
from scipy.spatial.distance import squareform

def hierarchical_clustering(data, title):
    fig, ax = plt.subplots(1, 1, figsize=(14, 8), dpi=120)
    correlations = data.corr()
    converted_corr = 1 - np.abs(correlations)
    Z = linkage(squareform(converted_corr), 'complete')

    dn = dendrogram(Z, labels=data.columns, ax=ax, above_threshold_color='#ff0000', orientation='right')
    hierarchy.set_link_color_palette(None)
    plt.grid(axis='x')
    plt.title(f'{title} Hierarchical clustering, Dendrogram', fontsize=18, fontweight='bold')
    plt.show()

hierarchical_clustering(df_train[num_cols+binary_cols], title='Train data')
hierarchical_clustering(df_test[num_cols+binary_cols], title='Test data')
```



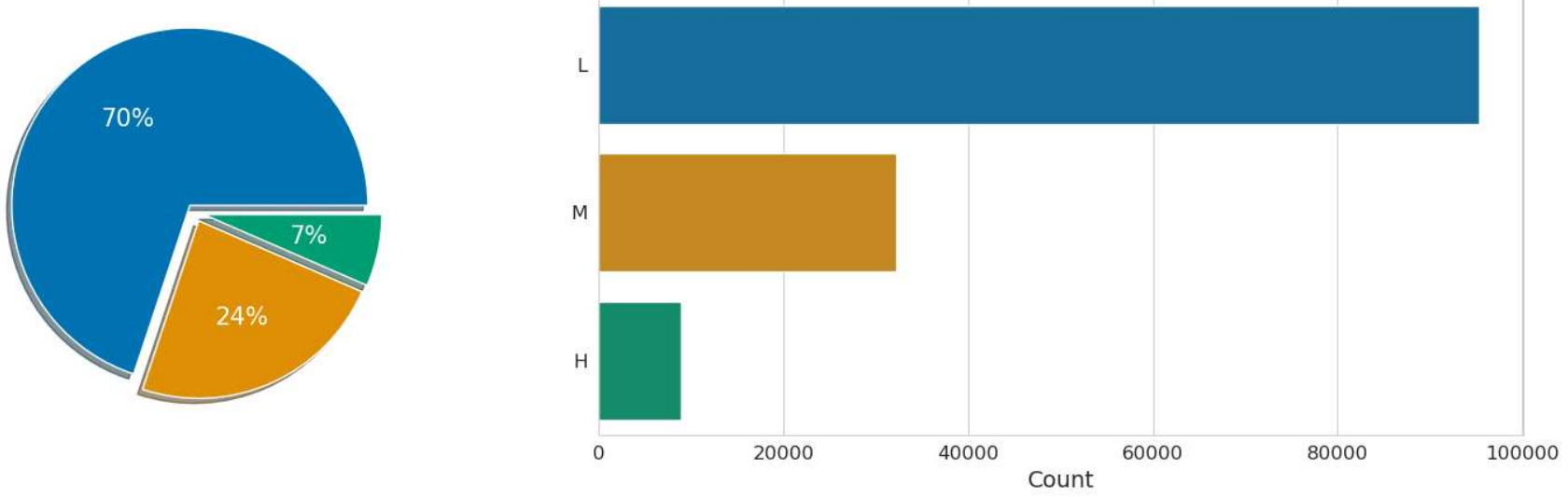


In [8]:

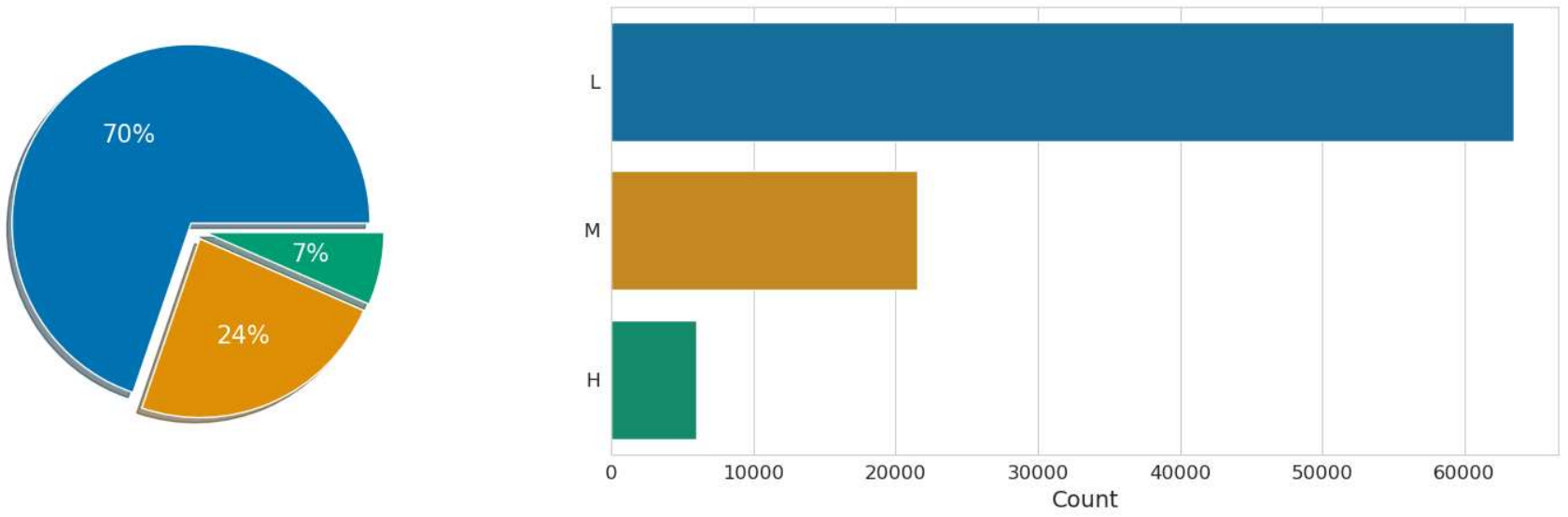
```python
def plot_target_feature(df_train, target_col, figsize=(16,5), palette='colorblind', name='Train'):
    df_train = df_train.fillna('Nan')

    fig, ax = plt.subplots(1, 2, figsize=figsize)
    ax = ax.flatten()

    # Pie chart
    pie_colors = sns.color_palette(palette, len(df_train[target_col].unique()))
    ax[0].pie(
        df_train[target_col].value_counts(),
        shadow=True,
        explode=[0.05] * len(df_train[target_col].unique()),
        autopct='%1.f%%',
        textprops={'size': 15, 'color': 'white'},
        colors=pie_colors
    )
    ax[0].set_aspect('equal')  # Fix the aspect ratio to make the pie chart circular

    # Bar plot
    bar_colors = sns.color_palette(palette)
    sns.countplot(
        data=df_train,
        y=target_col,
        ax=ax[1],
        palette=bar_colors
    )
    ax[1].set_xlabel('Count', fontsize=14)
    ax[1].set_ylabel('')
    ax[1].tick_params(labelsize=12)
    ax[1].yaxis.set_tick_params(width=0)  # Remove tick lines for y-axis

    fig.suptitle(f'{target_col} in {name} Dataset', fontsize=16, fontweight='bold')
    plt.tight_layout()

    # Show the plot
    plt.show()

plot_target_feature(df_train, 'Type', figsize=(16,5), palette='colorblind', name='Train data')
plot_target_feature(df_test, 'Type', figsize=(16,5), palette='colorblind', name='Test data')
```

```
plot_target_feature(df_train, target_col, figsize=(16,5), palette='colorblind', name='Train data')
plot_target_feature(original, target_col, figsize=(16,5), palette='colorblind', name='Original data')
```

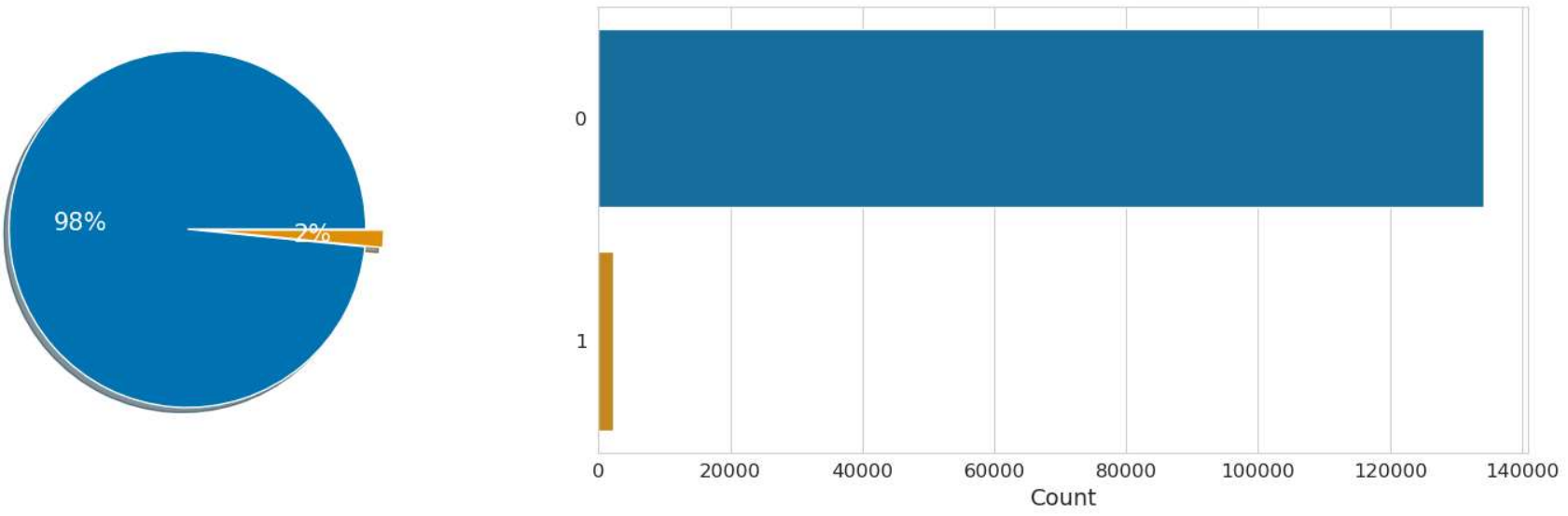
Type in Train data Dataset



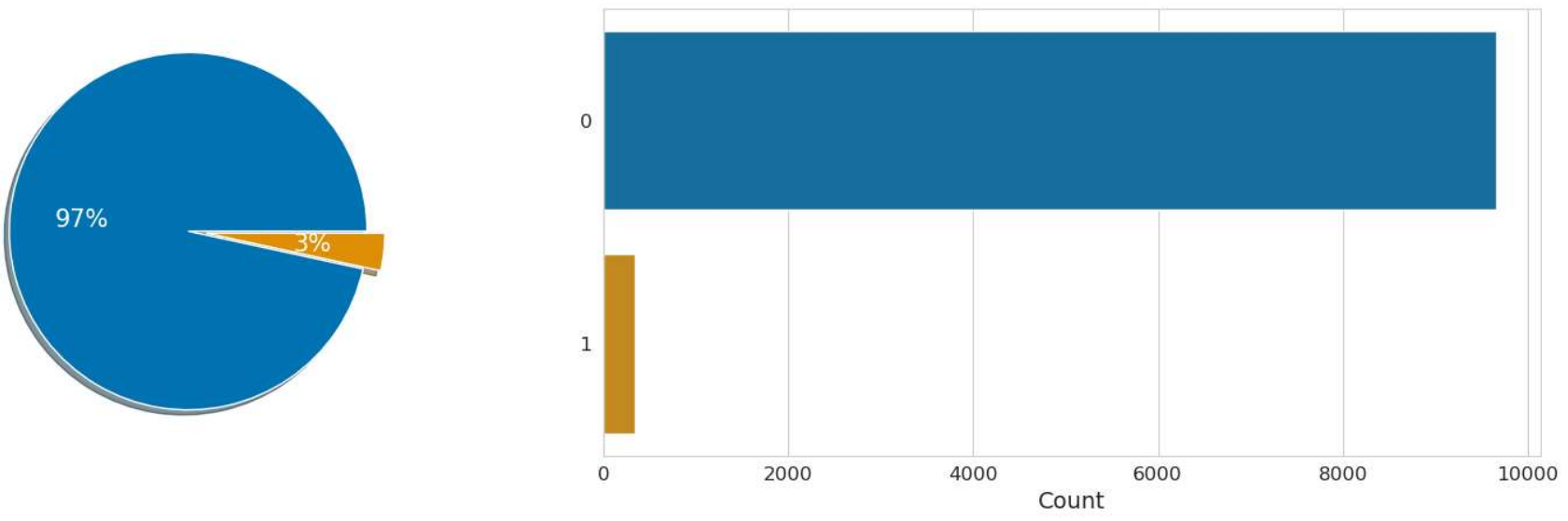
Type in Test data Dataset



Machine failure in Train data Dataset

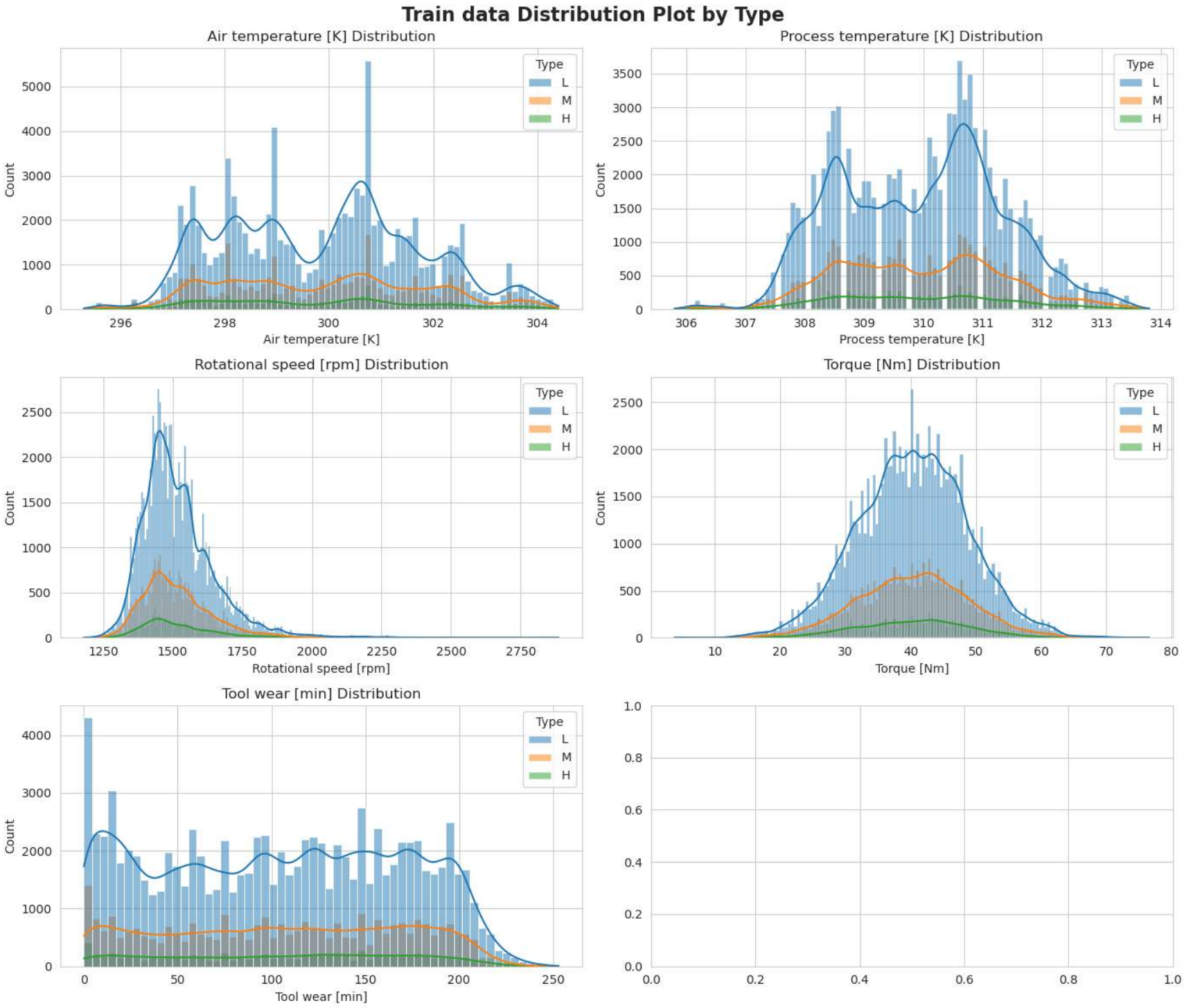


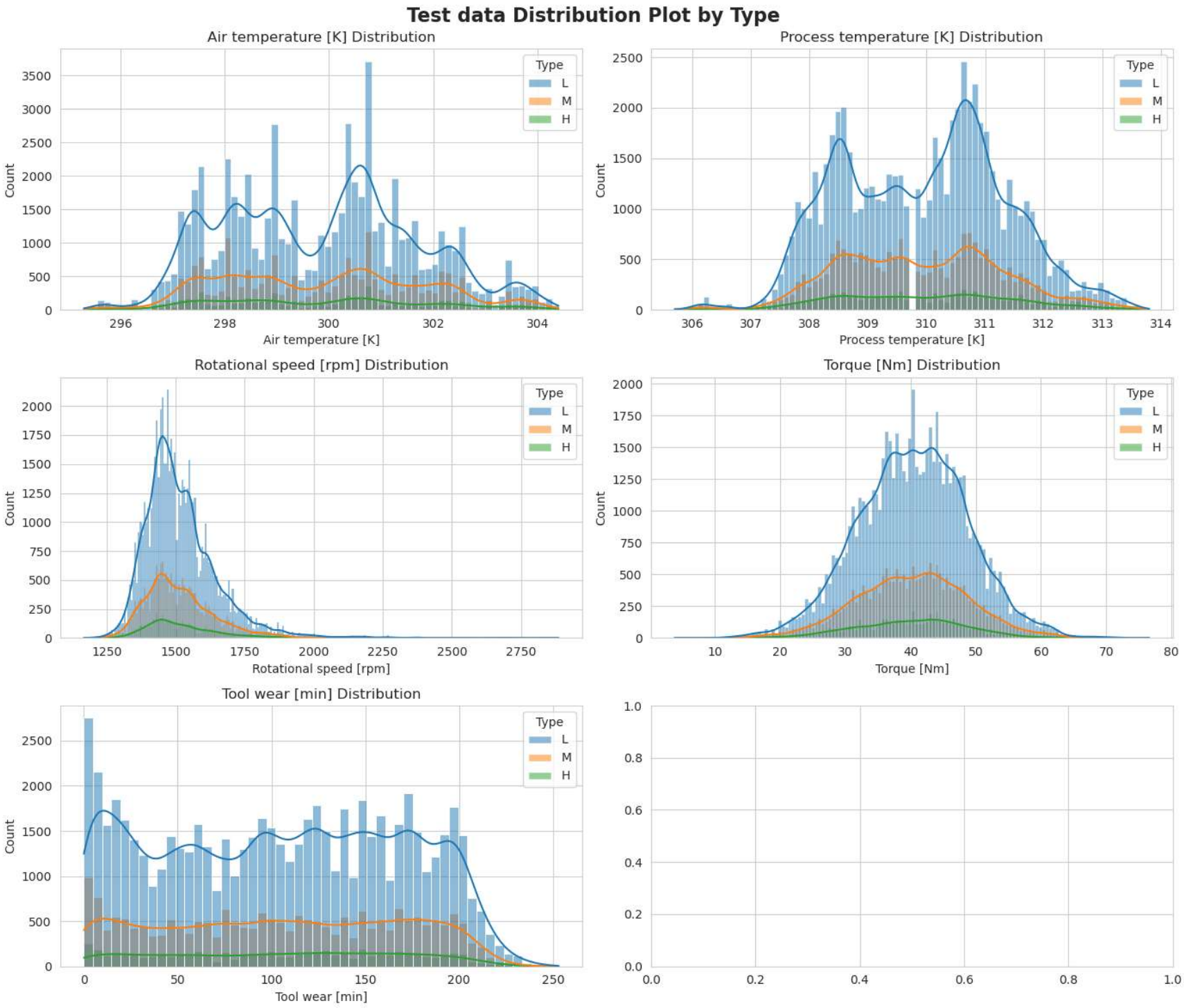
Machine failure in Original data Dataset



In [9]:
```
def plot_distribution(df, hue, title='', drop_cols=[]):
    sns.set_style('whitegrid')

    cols = df.columns.drop([hue] + drop_cols)
    n_cols = 2
    n_rows = (len(cols) - 1) // n_cols + 1

    fig, axes = plt.subplots(nrows=n_rows, ncols=n_cols, figsize=(14, 4*n_rows))

    for i, var_name in enumerate(cols):
        row = i // n_cols
        col = i % n_cols

        ax = axes[row, col]
        sns.histplot(data=df, x=var_name, kde=True, ax=ax, hue=hue) # sns.distplot(df_train[var_name], kde=True, ax=ax,
```

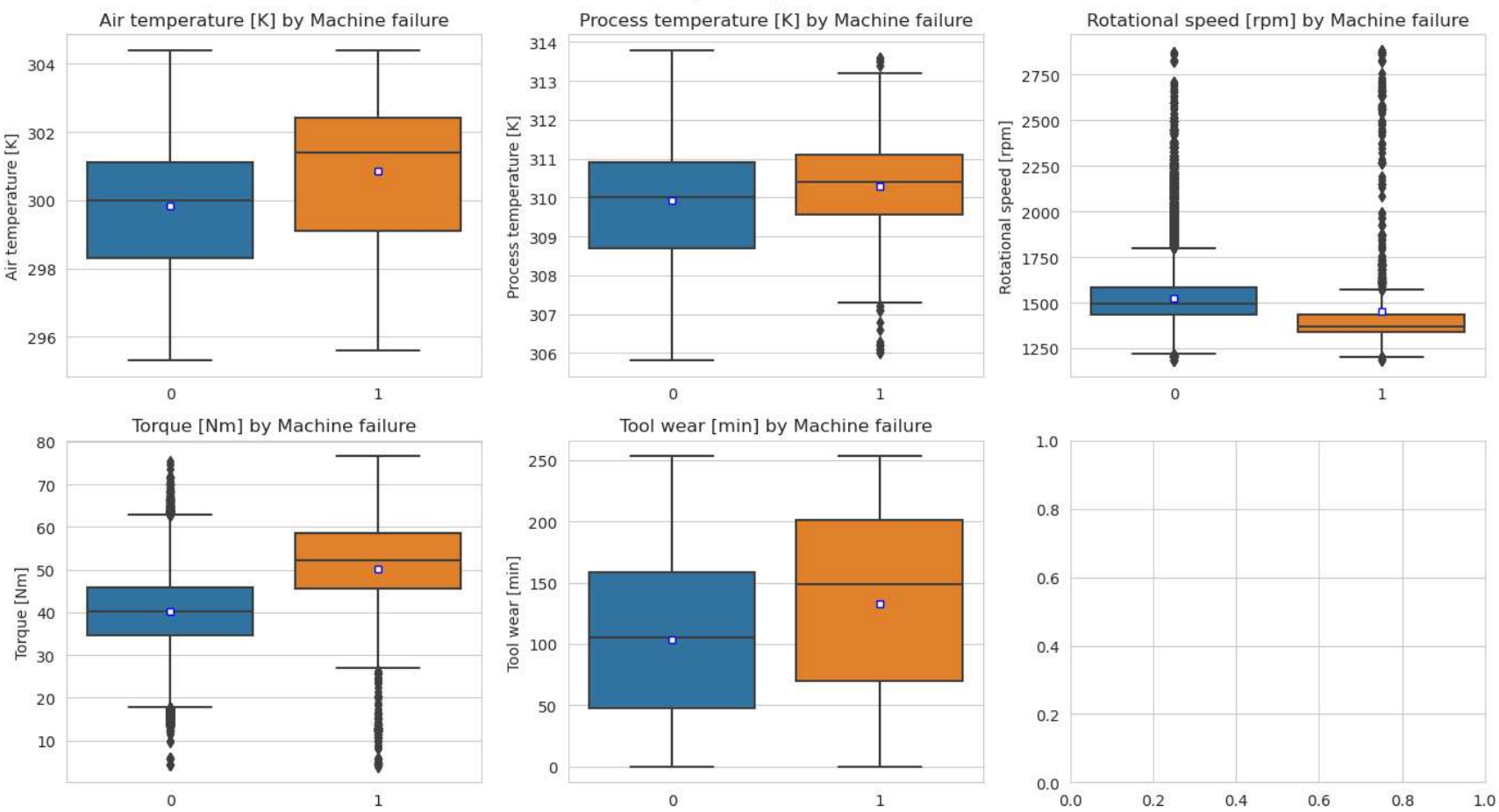
```
        ax.set_title(f'{var_name} Distribution')

    fig.suptitle(f'{title} Distribution Plot by {hue}', fontweight='bold', fontsize=16)
    plt.tight_layout()
    plt.show()

plot_distribution(df_train[num_cols+['Type']], hue='Type', title='Train data')
plot_distribution(df_test[num_cols+['Type']], hue='Type', title='Test data')
# plot_distribution(df_train[num_cols+[target_col]], hue=target_col, title='Train data')
# plot_distribution(original[num_cols+[target_col]], hue=target_col, title='Original data')
```



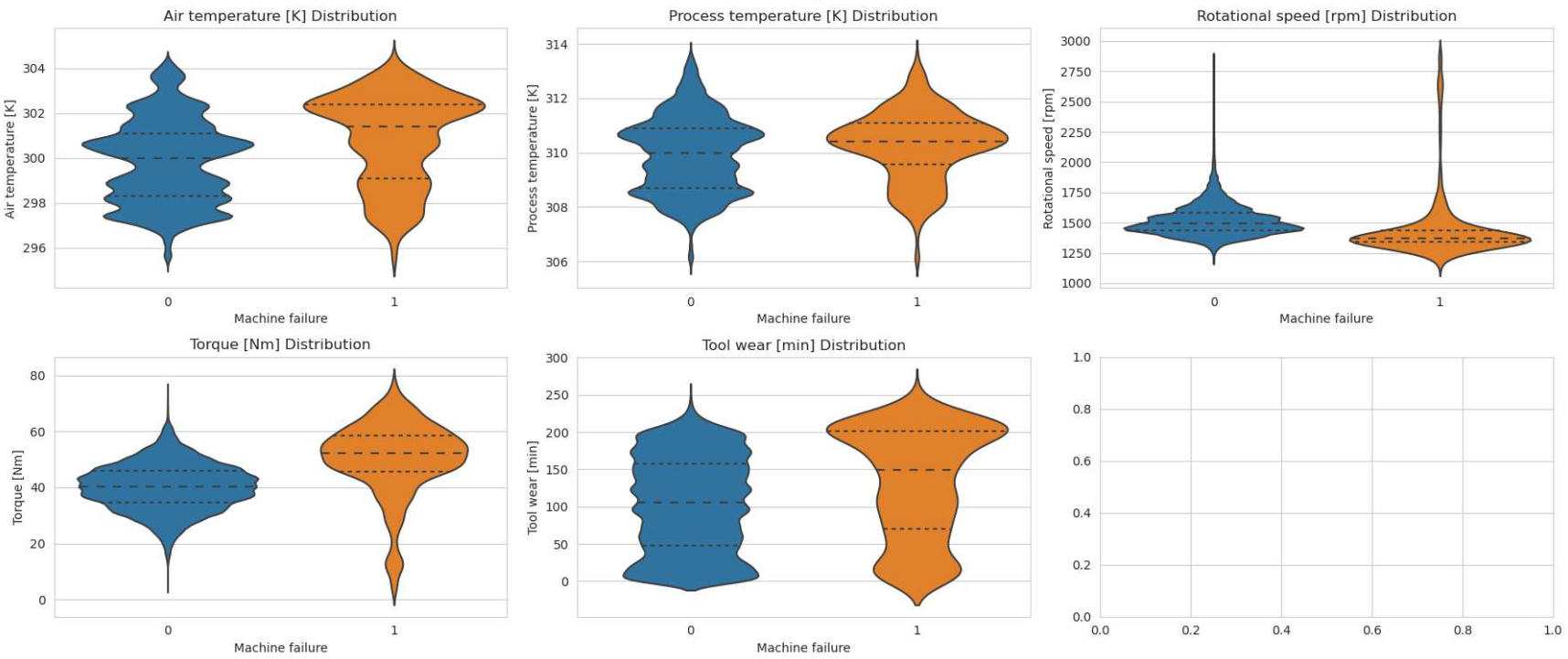


In [10]:

```python
def plot_boxplot(df, hue, drop_cols=[], n_cols=3, title=''):
    sns.set_style('whitegrid')

    cols = df.columns.drop([hue] + drop_cols)
    n_rows = (len(cols) - 1) // n_cols + 1

    fig, axes = plt.subplots(nrows=n_rows, ncols=n_cols, figsize=(14, 4*n_rows))

    for i, var_name in enumerate(cols):
        row = i // n_cols
        col = i % n_cols

        ax = axes[row, col]
        sns.boxplot(data=df, x=hue, y=var_name, ax=ax, showmeans=True,
                    meanprops={"marker":"s","markerfacecolor":"white", "markeredgecolor":"blue", "markersize":"5"})
        ax.set_title(f'{var_name} by {hue}')
        ax.set_xlabel('')

    fig.suptitle(f'{title} Boxplot by {hue}', fontweight='bold', fontsize=16)
    plt.tight_layout()
    plt.show()

plot_boxplot(df_train[num_cols+[target_col]], hue=target_col, n_cols=3, title='Train data')
plot_boxplot(original[num_cols+[target_col]], hue=target_col, n_cols=3, title='Original data')
```



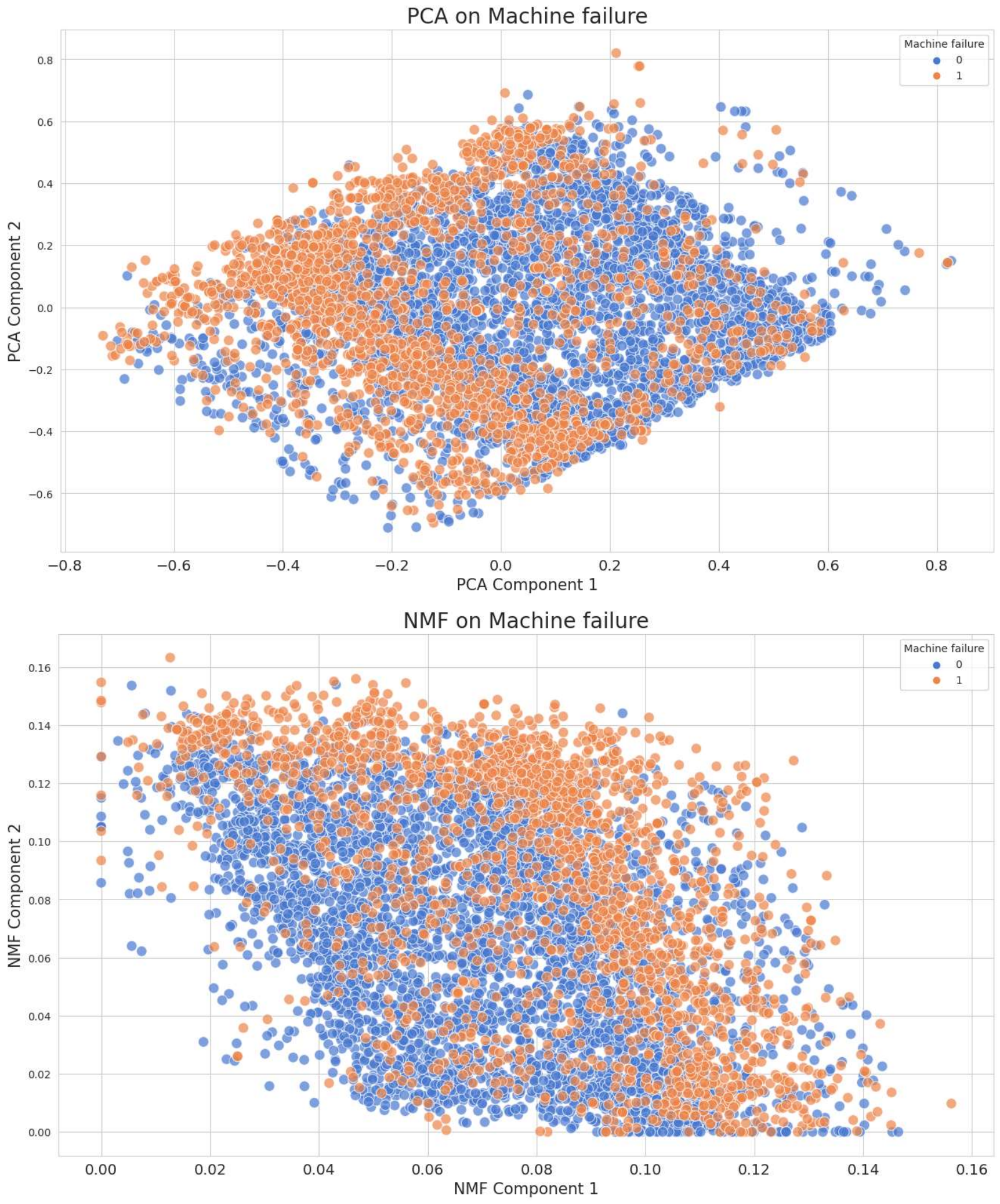
In [11]:
```python
def plot_violinplot(df, hue, drop_cols=[], n_cols=2, title=''):
    sns.set_style('whitegrid')

    cols = df.columns.drop([hue] + drop_cols)
    n_rows = (len(cols) - 1) // n_cols + 1

    fig, axes = plt.subplots(nrows=n_rows, ncols=n_cols, figsize=(18, 4*n_rows))

    for i, var_name in enumerate(cols):
        row = i // n_cols
        col = i % n_cols

        ax = axes[row, col]
        sns.violinplot(data=df, x=hue, y=var_name, ax=ax, inner='quartile')
        ax.set_title(f'{var_name} Distribution')

    fig.suptitle(f'{title} Violin Plot by {hue}', fontweight='bold', fontsize=16)
    plt.tight_layout()
    plt.show()

plot_violinplot(df_train[num_cols+[target_col]], hue=target_col, n_cols=3, title='Train data')
plot_violinplot(original[num_cols+[target_col]], hue=target_col, n_cols=3, title='Original data')
```



In [12]:

```python
class Decomp:
    def __init__(self, n_components, method="pca", scaler_method='standard'):
        self.n_components = n_components
        self.method = method
        self.scaler_method = scaler_method

    def dimension_reduction(self, df):

        X_reduced = self.dimension_method(df)
        df_comp = pd.DataFrame(X_reduced, columns=[f'{self.method.upper()}_{_}' for _ in range(self.n_components)], ind
        return df_comp

    def dimension_method(self, df):

        X = self.scaler(df)
        if self.method == "pca":
            pca = PCA(n_components=self.n_components, random_state=0)
            X_reduced = pca.fit_transform(X)
            self.comp = pca
        elif self.method == "nmf":
            nmf = NMF(n_components=self.n_components, random_state=0)
            X_reduced = nmf.fit_transform(X)
        else:
            raise ValueError(f"Invalid method name: {method}")

        return X_reduced

    def scaler(self, df):

        _df = df.copy()

        if self.scaler_method == "standard":
            return StandardScaler().fit_transform(_df)
        elif self.scaler_method == "minmax":
            return MinMaxScaler().fit_transform(_df)
        elif self.scaler_method == None:
            return _df.values
        else:
            raise ValueError(f"Invalid scaler_method name")

    def get_columns(self):
        return [f'{self.method.upper()}_{_}' for _ in range(self.n_components)]

    def get_explained_variance_ratio(self):
        return np.sum(self.comp.explained_variance_ratio_)

    def transform(self, df):
```

```
        X = self.scaler(df)
        X_reduced = self.comp.transform(X)
        df_comp = pd.DataFrame(X_reduced, columns=[f'{self.method.upper()}_{_}' for _ in range(self.n_components)], ind

        return df_comp

    def decomp_plot(self, tmp, label, hue='genre'):
        plt.figure(figsize = (16, 9))
        sns.scatterplot(x = f"{label}_0", y = f"{label}_1", data=tmp, hue=hue, alpha=0.7, s=100, palette='muted');

        plt.title(f'{label} on {hue}', fontsize = 20)
        plt.xticks(fontsize = 14)
        plt.yticks(fontsize = 10);
        plt.xlabel(f"{label} Component 1", fontsize = 15)
        plt.ylabel(f"{label} Component 2", fontsize = 15)


data = X_train_res[num_cols].copy()
for method in ['pca', 'nmf']:
    decomp = Decomp(n_components=2, method=method, scaler_method='minmax')
    decomp_feature = decomp.dimension_reduction(data)
    decomp_feature = pd.concat([y_train_res, decomp_feature], axis=1)
    decomp.decomp_plot(decomp_feature, method.upper(), target_col)

del X_train_res, y_train_res, rus, data
```



Feature Engineering

- `replace_Type()` : Perform label encoding for the `Type` column.

- `create_features()` : Create new features.
- `add_pca_features()` : Add PCA results as features.
- `cat_encoder` : Label Encoder

Note: Not all of the above feature engineering is adapted in this kernel. Please take it as an idea.

In [13]:
```python
def create_features(df):

    # Create a new feature by subtracting 'Air temperature' from 'Process temperature'
    # df['Temperature difference [K]'] = df['Process temperature [K]'] - df['Air temperature [K]']

    # Create a new feature by divided 'Air temperature' from 'Process temperature'
    df["Temperature ratio"] = df['Process temperature [K]'] / df['Air temperature [K]']

    # Create a new feature by multiplying 'Torque' and 'Rotational speed'
    df['Torque * Rotational speed'] = df['Torque [Nm]'] * df['Rotational speed [rpm]']

    # Create a new feature by multiplying 'Torque' by 'Tool wear'
    df['Torque * Tool wear'] = df['Torque [Nm]'] * df['Tool wear [min]']

    # Create a new feature by adding 'Air temperature' and 'Process temperature'
    # df['Temperature sum [K]'] = df['Air temperature [K]'] + df['Process temperature [K]']

    # Create a new feature by multiplying 'Torque' by 'Rotational speed'
    df['Torque * Rotational speed'] = df['Torque [Nm]'] * df['Rotational speed [rpm]']

    new_cols = [
        #'Temperature difference [K]',
        'Temperature ratio',
        'Torque * Rotational speed',
        'Torque * Tool wear',
        # 'Temperature sum [K]',
        'Torque * Rotational speed'
    ]

    return df, new_cols

def add_pca_features(X_train, X_test):

    # Select the columns for PCA
    pca_features = X_train.select_dtypes(include=['float64']).columns.tolist()
    n_components = 2 # len(pca_features)

    # Create the pipeline
    pipeline = make_pipeline(StandardScaler(), PCA(n_components=n_components))

    # Perform PCA
    pipeline.fit(X_train[pca_features])

    # Create column names for PCA features
    pca_columns = [f'PCA_{i}' for i in range(n_components)]

    # Add PCA features to the dataframe
    X_train[pca_columns] = pipeline.transform(X_train[pca_features])
    X_test[pca_columns] = pipeline.transform(X_test[pca_features])

    return X_train, X_test

def replace_Type(df):

    df["Type"] = df["Type"].replace({'L':0})
    df["Type"] = df["Type"].replace({'M':1})
    df["Type"] = df["Type"].replace({'H':2})
    df["Type"] = df["Type"].astype(int)

    return df

def cat_encoder(X_train, X_test, cat_cols, encode='label'):

    if encode == 'label':
        ## Label Encoder
        encoder = OrdinalEncoder(cols=cat_cols)
        train_encoder = encoder.fit_transform(X_train[cat_cols]).astype(int)
        test_encoder = encoder.transform(X_test[cat_cols]).astype(int)
        X_train[cat_cols] = train_encoder[cat_cols]
        X_test[cat_cols] = test_encoder[cat_cols]
        encoder_cols = cat_cols

    else:
        ## OneHot Encoder
        encoder = OneHotEncoder(handle_unknown='ignore', sparse_output=False)
        train_encoder = encoder.fit_transform(X_train[cat_cols]).astype(int)
        test_encoder = encoder.transform(X_test[cat_cols]).astype(int)
        X_train = pd.concat([X_train, train_encoder], axis=1)
        X_test = pd.concat([X_test, test_encoder], axis=1)
        X_train.drop(cat_cols, axis=1, inplace=True)
        X_test.drop(cat_cols, axis=1, inplace=True)
        encoder_cols = list(train_encoder.columns)

    return X_train, X_test, encoder_cols

def rename(df):
    df.columns = df.columns.str.replace('\[.*?\]', '', regex=True)
    df.columns = df.columns.str.strip()
```

```
    # df.columns = df.columns.str.replace(' ', '')

    return df
```

In [14]:
```
# Concatenate train and original dataframes, and prepare train and test sets
train = pd.concat([df_train, original])
test = df_test.copy()

X_train = train.drop([f'{target_col}'],axis=1).reset_index(drop=True)
y_train = train[f'{target_col}'].reset_index(drop=True)
X_test = test.reset_index(drop=True)

# Category Encoders
X_train = replace_Type(X_train)
X_test = replace_Type(X_test)
X_train, X_test, _ = cat_encoder(X_train, X_test, ['Product ID'], encode='label')
cat_cols = ['Type', 'Product ID']

# Create Features
new_cols = []
X_train, _ = create_features(X_train)
X_test, new_cols = create_features(X_test)
# X_train, X_test = add_pca_features(X_train, X_test)

# StandardScaler
sc = StandardScaler() # MinMaxScaler or StandardScaler
X_train[num_cols+new_cols] = sc.fit_transform(X_train[num_cols+new_cols])
X_test[num_cols+new_cols] = sc.transform(X_test[num_cols+new_cols])

# Drop_col
drop_cols = ['is_generated', 'RNF'] # binary_cols
X_train.drop(drop_cols, axis=1, inplace=True)
X_test.drop(drop_cols, axis=1, inplace=True)

# Rename
X_train = rename(X_train)
X_test = rename(X_test)

print(f"X_train shape :{X_train.shape} , y_train shape :{y_train.shape}")
print(f"X_test shape :{X_test.shape}")

del train, test, df_train, df_test

X_train.head(5)
```

```
X_train shape :(146429, 14) , y_train shape :(146429,)
X_test shape :(90954, 14)
```

Out[14]:

	Product ID	Type	Air temperature	Process temperature	Rotational speed	Torque	Tool wear	TWF	HDF	PWF	OSF	Temperature ratio	Torque * Rotational speed	Torque * Tool wear
0	1	0	0.388563	-0.248148	0.524194	-0.490542	0.552766	0	0	0	0	-1.068286	-0.289927	0.300523
1	2	1	1.456753	1.547557	1.672486	-1.303478	1.491004	0	0	0	0	-0.644474	-0.961066	0.575923
2	3	0	-0.305761	-1.038258	1.996544	-1.605426	-1.245523	0	0	0	0	-0.835680	-1.311256	-1.278352
3	4	0	0.602201	0.685618	0.016972	0.461755	1.444092	0	0	0	0	-0.208621	0.743354	1.621112
4	5	1	-1.000084	-0.679117	0.841207	-0.571836	-1.104788	0	0	0	0	0.963453	-0.240255	-1.083811

Data Splitting

In [15]:
```
class Splitter:
    def __init__(self, kfold=True, n_splits=5, cat_df=pd.DataFrame(), test_size=0.5):
        self.n_splits = n_splits
        self.kfold = kfold
        self.cat_df = cat_df
        self.test_size = test_size

    def split_data(self, X, y, random_state_list):
        if self.kfold == 'skf':
            for random_state in random_state_list:
                kf = StratifiedKFold(n_splits=self.n_splits, random_state=random_state, shuffle=True)
                for train_index, val_index in kf.split(X, self.cat_df):
                    X_train, X_val = X.iloc[train_index], X.iloc[val_index]
                    y_train, y_val = y.iloc[train_index], y.iloc[val_index]
                    yield X_train, X_val, y_train, y_val, val_index
        elif self.kfold:
            for random_state in random_state_list:
                kf = KFold(n_splits=self.n_splits, random_state=random_state, shuffle=True)
                for train_index, val_index in kf.split(X, y):
                    X_train, X_val = X.iloc[train_index], X.iloc[val_index]
                    y_train, y_val = y.iloc[train_index], y.iloc[val_index]
                    yield X_train, X_val, y_train, y_val, val_index
        else:
            for random_state in random_state_list:
                X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=self.test_size, random_state=random_s
                yield X_train, X_val, y_train, y_val
```

Define Model

LightGBM, CatBoost Xgboost and HistGradientBoosting hyper parameters are determined by optuna.

In [16]:
```
class Classifier:
    def __init__(self, n_estimators=100, device="cpu", random_state=0):
        self.n_estimators = n_estimators
        self.device = device
        self.random_state = random_state
        self.models = self._define_model()
        self.models_name = list(self._define_model().keys())
        self.len_models = len(self.models)

    def _define_model(self):

        xgb1_params = {
            'n_estimators': self.n_estimators,
            'learning_rate': 0.0503196477566407,
            'booster': 'gbtree',
            'lambda': 0.00379319640405843,
            'alpha': 0.106754104302093,
            'subsample': 0.938028434508189,
            'colsample_bytree': 0.212545425027345,
            'max_depth': 9,
            'min_child_weight': 2,
            'eta': 1.03662446190642E-07,
            'gamma': 0.000063826049787043,
            'grow_policy': 'lossguide',
            'n_jobs': -1,
            'objective': 'binary:logistic',
            #'eval_metric': 'auc',
            'verbosity': 0,
            'random_state': self.random_state,
        }
        xgb2_params = {
            'n_estimators': self.n_estimators,
            'learning_rate': 0.00282353606391198,
            'booster': 'gbtree',
            'lambda': 0.399776698351379,
            'alpha': 1.01836149061356E-07,
            'subsample': 0.957123754766769,
            'colsample_bytree': 0.229857555596548,
            'max_depth': 9,
            'min_child_weight': 4,
            'eta': 2.10637756839133E-07,
            'gamma': 0.00314857715085414,
            'grow_policy': 'depthwise',
            'n_jobs': -1,
            'objective': 'binary:logistic',
            #'eval_metric': 'auc',
            'verbosity': 0,
            'random_state': self.random_state,
        }
        xgb3_params = {
            'n_estimators': self.n_estimators,
            'learning_rate': 0.00349356650247156,
            'booster': 'gbtree',
            'lambda': 0.0002963239871324443,
            'alpha': 0.0000162103492458353,
            'subsample': 0.822994064549709,
            'colsample_bytree': 0.244618079894501,
            'max_depth': 9,
            'min_child_weight': 2,
            'eta': 8.03406601824666E-06,
            'gamma': 3.91180893163099E-07,
            'grow_policy': 'depthwise',
            'n_jobs': -1,
            'objective': 'binary:logistic',
            #'eval_metric': 'auc',
            'verbosity': 0,
            'random_state': self.random_state,
        }
        if self.device == 'gpu':
            xgb_params['tree_method'] = 'gpu_hist'
            xgb_params['predictor'] = 'gpu_predictor'

        lgb1_params = {
            'n_estimators': self.n_estimators,
            'learning_rate': 0.0124415817896377,
            'reg_alpha': 0.00139174509988134,
            'reg_lambda': 0.000178964551019674,
            'num_leaves': 249,
            'colsample_bytree': 0.675264038614975,
            'subsample': 0.421482143660471,
            'subsample_freq': 4,
            'min_child_samples': 8,
            'objective': 'binary',
            'metric': 'binary_error',
            'boosting_type': 'gbdt',
            'is_unbalance':True,
            # 'n_jobs': -1,
            #'force_row_wise': True,
            'device': self.device,
            'random_state': self.random_state
        }
        lgb2_params = {
            'n_estimators': self.n_estimators,
```

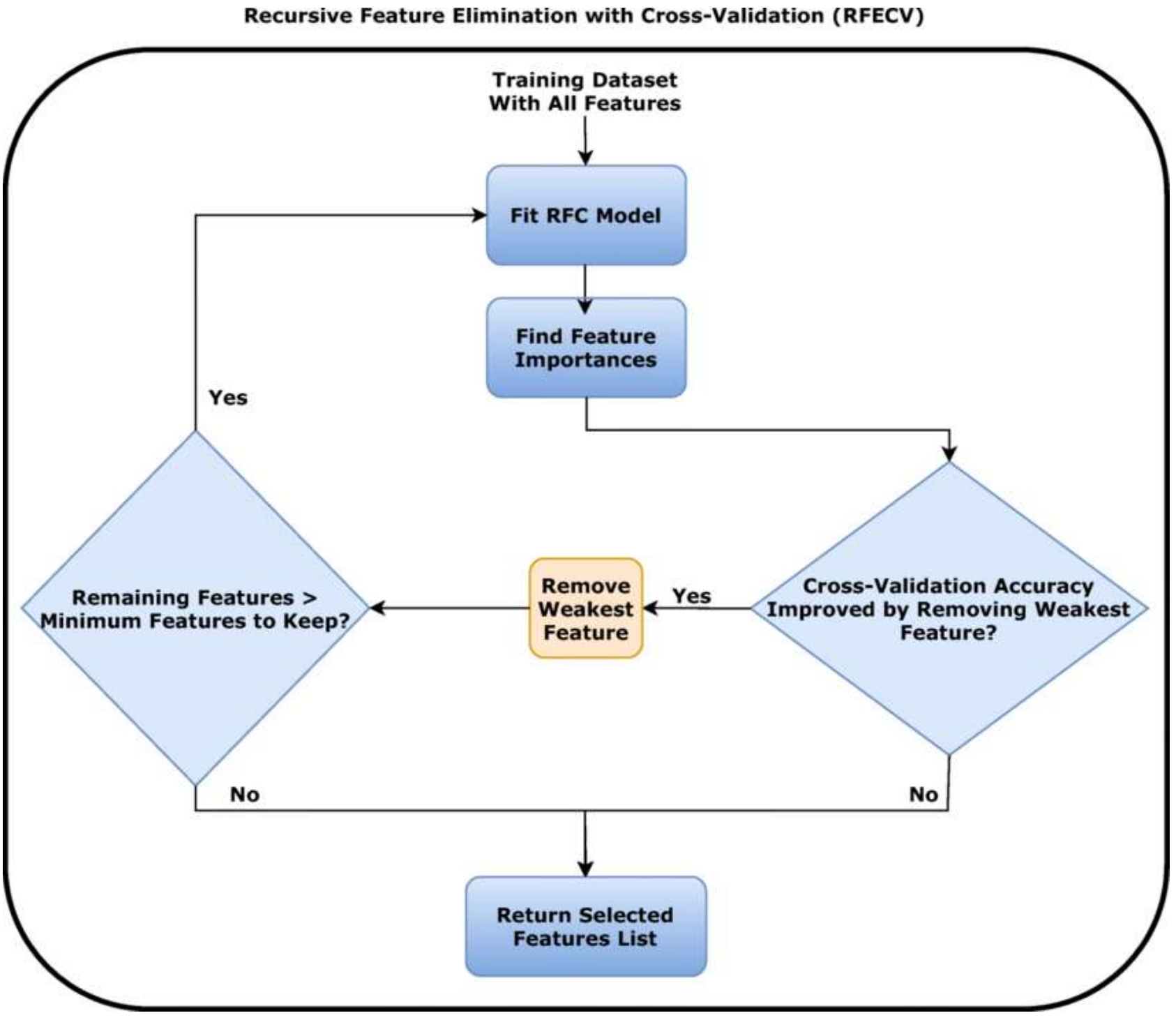
```
            'learning_rate': 0.0247403801218241,
            'reg_alpha': 6.84813726047269E-06,
            'reg_lambda': 3.40443691552308E-08,
            'num_leaves': 223,
            'colsample_bytree': 0.597332047776164,
            'subsample': 0.466442641250326,
            'subsample_freq': 2,
            'min_child_samples': 5,
            'objective': 'binary',
            'metric': 'binary_error',
            'boosting_type': 'gbdt',
            'is_unbalance':True,
            # 'n_jobs': -1,
            #'force_row_wise': True,
            'device': self.device,
            'random_state': self.random_state
        }
        lgb3_params = {
            'n_estimators': self.n_estimators,
            'learning_rate': 0.0109757020463629,
            'reg_alpha': 0.174927073496136,
            'reg_lambda': 2.45325882544558E-07,
            'num_leaves': 235,
            'colsample_bytree': 0.756605772162953,
            'subsample': 0.703911560320816,
            'subsample_freq': 5,
            'min_child_samples': 21,
            'objective': 'binary',
            'metric': 'binary_error',
            'boosting_type': 'gbdt',
            'is_unbalance':True,
            # 'n_jobs': -1,
            #'force_row_wise': True,
            'device': self.device,
            'random_state': self.random_state
        }
        cat1_params = {
            'iterations': self.n_estimators,
            'depth': 3,
            'learning_rate': 0.020258010893459,
            'l2_leaf_reg': 0.583685138705941,
            'random_strength': 0.177768021213223,
            'od_type': "Iter",
            'od_wait': 116,
            'bootstrap_type': "Bayesian",
            'grow_policy': 'Depthwise',
            'bagging_temperature': 0.478048798393903,
            'eval_metric': 'Logloss', # AUC
            'loss_function': 'Logloss',
            'auto_class_weights': 'Balanced',
            'task_type': self.device.upper(),
            'verbose': False,
            'allow_writing_files': False,
            'random_state': self.random_state
        }
        cat2_params = {
            'iterations': self.n_estimators,
            'depth': 5,
            'learning_rate': 0.00666304601039438,
            'l2_leaf_reg': 0.0567881687170355,
            'random_strength': 0.00564702921370138,
            'od_type': "Iter",
            'od_wait': 93,
            'bootstrap_type': "Bayesian",
            'grow_policy': 'Depthwise',
            'bagging_temperature': 2.48298505165348,
            'eval_metric': 'Logloss', # AUC
            'loss_function': 'Logloss',
            'auto_class_weights': 'Balanced',
            'task_type': self.device.upper(),
            'verbose': False,
            'allow_writing_files': False,
            'random_state': self.random_state
        }
        cat3_params = {
            'iterations': self.n_estimators,
            'depth': 5,
            'learning_rate': 0.0135730417743519,
            'l2_leaf_reg': 0.0597353604503262,
            'random_strength': 0.0675876600077264,
            'od_type': "Iter",
            'od_wait': 122,
            'bootstrap_type': "Bayesian",
            'grow_policy': 'Depthwise',
            'bagging_temperature': 1.85898154006468,
            'eval_metric': 'Logloss', # AUC
            'loss_function': 'Logloss',
            'auto_class_weights': 'Balanced',
            'task_type': self.device.upper(),
            'verbose': False,
            'allow_writing_files': False,
            'random_state': self.random_state
        }
        hist_params = {
            'l2_regularization': 0.880153002159043,
```

```
            'learning_rate': 0.00251637614495506,
            'max_iter': self.n_estimators,
            'max_depth': 18,
            'max_bins': 255,
            'min_samples_leaf': 67,
            'max_leaf_nodes':66,
            'early_stopping': True,
            'n_iter_no_change': 50,
            'categorical_features': ['Type'],
            'class_weight':'balanced',
            'random_state': self.random_state
        }

        models = {
            "xgb": xgb.XGBClassifier(**xgb1_params),
            #"xgb2": xgb.XGBClassifier(**xgb2_params),
            #"xgb3": xgb.XGBClassifier(**xgb3_params),
            "lgb": lgb.LGBMClassifier(**lgb1_params),
            "lgb2": lgb.LGBMClassifier(**lgb2_params),
            "lgb3": lgb.LGBMClassifier(**lgb3_params),
            "cat": CatBoostClassifier(**cat1_params),
            "cat2": CatBoostClassifier(**cat2_params),
            "cat3": CatBoostClassifier(**cat3_params),
            'hgb': HistGradientBoostingClassifier(**hist_params),
            'rf': RandomForestClassifier(n_estimators=500, n_jobs=-1, class_weight="balanced", random_state=self.random
            #'brf': BalancedRandomForestClassifier(n_estimators=1000, random_state=self.random_state), #  n_jobs=-1,
            'lr': LogisticRegressionCV(max_iter=2000, random_state=self.random_state),
            #'svc': SVC(max_iter=300, kernel="rbf", gamma="auto", probability=True, class_weight="balanced",random_state
        }

        return models
```

Feature Selection (RFE-CV)

RFECV is a technique for automated feature selection that combines recursive feature elimination and cross-validation to identify the optimal subset of features for a given machine learning task.

Note: RFE-CV takes a lot of time. Here n_estimators are reduced to save time. When originally used, it is recommended to run with the actual hyperparameters.



In [17]:
```
splitter = Splitter(kfold=False, test_size=0.6)
for X_train_, X_val, y_train_, y_val in splitter.split_data(X_train, y_train, random_state_list=[42]):
    print('Data set by train_test_split')
```

```
Data set by train_test_split
```

In [18]:
```
%%time

n_estimators = 200
scoring = 'roc_auc'
min_features_to_select = 10

classifier = Classifier(n_estimators, device='cpu', random_state=0)
models = classifier.models

models_name = [_ for _ in classifier.models_name if ('xgb' in _) or ('lgb' in _) or ('cat' in _)]
trained_models = dict(zip(models_name, ['' for _ in range(classifier.len_models)]))
unnecessary_features = dict(zip(models_name, [[] for _ in range(classifier.len_models)]))
for name, model in models.items():
    if ('xgb' in name) or ('lgb' in name) or ('cat' in name):
        elimination = RFECV(
            model,
            step=1,
            min_features_to_select=min_features_to_select,
            cv=2,
            scoring=scoring,
            n_jobs=-1)
        elimination.fit(X_train_, y_train_)
        unnecessary_feature = list(X_train.columns[~elimination.get_support()])
        idx = np.argmax(elimination.cv_results_['mean_test_score'])
        mean_score = elimination.cv_results_['mean_test_score'][idx]
        std_score = elimination.cv_results_['std_test_score'][idx]
        print(f'{blu}{name}{res} {red} Best Mean{res} {scoring} {red}{mean_score:.5f} ± {std_score:.5f}{res} | N_STEP {
        print(f"Best unnecessary_feature: {unnecessary_feature}")
        removed_features = [f for i, f in enumerate(X_train.columns) if elimination.support_[i] == False]
        ranked_features = sorted(zip(X_train.columns, elimination.ranking_), key=lambda x: x[1])
        removed_features_by_ranking = [f[0] for f in ranked_features if f[0] in removed_features][::-1]
        print("Removed features:", removed_features_by_ranking)
        print(f'{"-" * 60}')

        trained_models[f'{name}'] = deepcopy(elimination)
        unnecessary_features[f'{name}'].extend(unnecessary_feature)

unnecessary_features = np.concatenate([_ for _ in unnecessary_features.values()])
features = np.unique(unnecessary_features, return_counts=True)[0]
counts = np.unique(unnecessary_features, return_counts=True)[1]
drop_features = list(features[counts >= 2])
print("Features recommended to be removed:", drop_features)
```

```
xgb  Best Mean roc_auc 0.96958 ± 0.00460 | N_STEP 3
Best unnecessary_feature: ['Type']
Removed features: ['Type']
------------------------------------------------------------
lgb  Best Mean roc_auc 0.96346 ± 0.00278 | N_STEP 3
Best unnecessary_feature: ['Type']
Removed features: ['Type']
------------------------------------------------------------
lgb2  Best Mean roc_auc 0.96526 ± 0.00157 | N_STEP 3
Best unnecessary_feature: ['Type']
Removed features: ['Type']
------------------------------------------------------------
lgb3  Best Mean roc_auc 0.96408 ± 0.00349 | N_STEP 4
Best unnecessary_feature: []
Removed features: []
------------------------------------------------------------
cat  Best Mean roc_auc 0.96537 ± 0.00457 | N_STEP 4
Best unnecessary_feature: []
Removed features: []
------------------------------------------------------------
cat2  Best Mean roc_auc 0.96354 ± 0.00453 | N_STEP 2
Best unnecessary_feature: ['Product ID', 'Type']
Removed features: ['Type', 'Product ID']
------------------------------------------------------------
cat3  Best Mean roc_auc 0.96423 ± 0.00386 | N_STEP 3
Best unnecessary_feature: ['Type']
Removed features: ['Type']
------------------------------------------------------------
Features recommended to be removed: ['Type']
CPU times: user 2min 43s, sys: 5.42 s, total: 2min 49s
Wall time: 3min 58s
```

In [ ]:
```
def plot_recursive_feature_elimination(elimination, scoring, min_features_to_select, name):
    n_scores = len(elimination.cv_results_["mean_test_score"])
    plt.figure(figsize=(10, 4))
    plt.xlabel("Number of features selected")
    plt.ylabel(f"{scoring}")

    # Plot the mean test scores with error bars
    plt.errorbar(
        range(min_features_to_select, n_scores + min_features_to_select),
        elimination.cv_results_["mean_test_score"],
        yerr=elimination.cv_results_["std_test_score"],
        fmt='o-',
        capsize=3,
        markersize=4,
    )

    plt.title(f"{name} Recursive Feature Elimination with correlated features", fontweight='bold')
    plt.grid(True)
    plt.tight_layout()
    plt.show()
```

```
for name, elimination in trained_models.items():
    plot_recursive_feature_elimination(elimination, scoring, min_features_to_select, name)
```

Configuration

In [ ]:
```
# Settings
kfold = 'skf'
n_splits = 10 # 10
n_reapts = 1 # 1
random_state = 42
n_estimators = 9999 # 99999
early_stopping_rounds = 200
n_trials = 2000 # 2000
verbose = False
device = 'cpu'

# Under Sampling
n_under_sampling = False # or False

# Pseudo Labeling
true_th = 0.99
false_th = 0.9999

# Fix seed
random.seed(random_state)
random_state_list = random.sample(range(9999), n_reapts)

# metrics
def auc(y_true, y_pred):
    return roc_auc_score(y_true, y_pred)
metric = auc
metric_name = metric.__name__.upper()

# To calculate runtime
def sec_to_minsec(t):
    min_ = int(t / 60)
    sec = int(t - min_*60)
    return min_, sec

# Process
```

One Model Xgboost

In [ ]:
```
feature_importances_ = pd.DataFrame(index=X_train.columns)
eval_results_ = {}
models_ = []
oof = np.zeros((X_train.shape[0]))
test_predss = np.zeros((X_test.shape[0]))

splitter = Splitter(kfold=kfold, n_splits=n_splits, cat_df=y_train)
for i, (X_train_, X_val, y_train_, y_val, val_index) in enumerate(splitter.split_data(X_train, y_train, random_state_li
    fold = i % n_splits
    m = i // n_splits

    # XGB .train() requires xgboost.DMatrix.
    # https://xgboost.readthedocs.io/en/stable/python/python_api.html#xgboost.DMatrix
    fit_set = xgb.DMatrix(X_train_, y_train_)
    val_set = xgb.DMatrix(X_val, y_val)
    watchlist = [(fit_set, 'fit'), (val_set, 'val')]

    # Training.
    # https://xgboost.readthedocs.io/en/stable/python/python_api.html#module-xgboost.training
    classifier = Classifier(3000, device)
    xgb_params = classifier.models['xgb'].get_params()
    # xgb_params = xgb.XGBClassifier(n_estimators=3000, learning_rate=0.01).get_params()

    eval_results_[fold] = {}
    model = xgb.train(
        num_boost_round=xgb_params['n_estimators'],
        params=xgb_params,
        dtrain=fit_set,
        evals=watchlist,
        evals_result=eval_results_[fold],
        verbose_eval=False,
        callbacks=[EarlyStopping(early_stopping_rounds, data_name='val', save_best=True)])

    val_preds = model.predict(val_set)
    test_predss += model.predict(xgb.DMatrix(X_test)) / n_splits

    oof[val_index] = val_preds

    val_score = metric(y_val, val_preds)
    best_iter = model.best_iteration
    print(f'Fold: {blu}{fold:>3}{res}| {metric_name}: {blu}{val_score:.5f}{res}' f' | Best iteration: {blu}{best_iter:>

    # Stores the feature importances
    feature_importances_[f'gain_{fold}'] = feature_importances_.index.map(model.get_score(importance_type='gain'))
    feature_importances_[f'split_{fold}'] = feature_importances_.index.map(model.get_score(importance_type='weight'))

    # Stores the model
    models_.append(model)
```

```
# Submission
sub = pd.read_csv(os.path.join(filepath, 'sample_submission.csv'))
sub[f'{target_col}'] = test_predss
sub.to_csv(f'xgb_submission.csv', index=False)

mean_cv_score_full = metric(y_train, oof)
print(f'{"*" * 50}\n{red}Mean{res} {metric_name} : {red}{mean_cv_score_full:.5f}')
```

In [ ]:

```
metric_score_folds = pd.DataFrame.from_dict(eval_results_).T
fit_rmsle = metric_score_folds.fit.apply(lambda x: x['logloss'])
val_rmsle = metric_score_folds.val.apply(lambda x: x['logloss'])

n_splits = len(metric_score_folds)
n_rows = math.ceil(n_splits / 3)

fig, axes = plt.subplots(n_rows, 3, figsize=(20, n_rows * 4), dpi=150)
ax = axes.flatten()

for i, (f, v, m) in enumerate(zip(fit_rmsle, val_rmsle, models_)):
    sns.lineplot(f, color='#B90000', ax=ax[i], label='fit')
    sns.lineplot(v, color='#048BA8', ax=ax[i], label='val')
    ax[i].legend()
    ax[i].spines['top'].set_visible(False)
    ax[i].spines['right'].set_visible(False)
    ax[i].set_title(f'Fold {i}', fontdict={'fontweight': 'bold'})

    color = ['#048BA8', '#90A6B1']
    best_iter = m.best_iteration
    span_range = [[0, best_iter], [best_iter + 10, best_iter + early_stopping_rounds]]

    for idx, sub_title in enumerate([f'Best\nIteration: {best_iter}', f'Early\n Stopping: {early_stopping_rounds}']):
        ax[i].annotate(sub_title,
                       xy=(sum(span_range[idx]) / 2, 4000),
                       xytext=(0, 0),
                       textcoords='offset points',
                       va="center",
                       ha="center",
                       color="w",
                       fontsize=12,
                       fontweight='bold',
                       bbox=dict(boxstyle='round4', pad=0.4, color=color[idx], alpha=0.6))
        ax[i].axvspan(span_range[idx][0] - 0.4, span_range[idx][1] + 0.4, color=color[idx], alpha=0.07)

    ax[i].set_xlim(0, best_iter + 20 + early_stopping_rounds)
    ax[i].set_xlabel('Boosting Round', fontsize=12)
    ax[i].set_ylabel('MAE', fontsize=12)
    ax[i].legend(loc='upper right', title=metric_name)

for j in range(i+1, n_rows * 3):
    ax[j].axis('off')

plt.tight_layout()
plt.show()
```

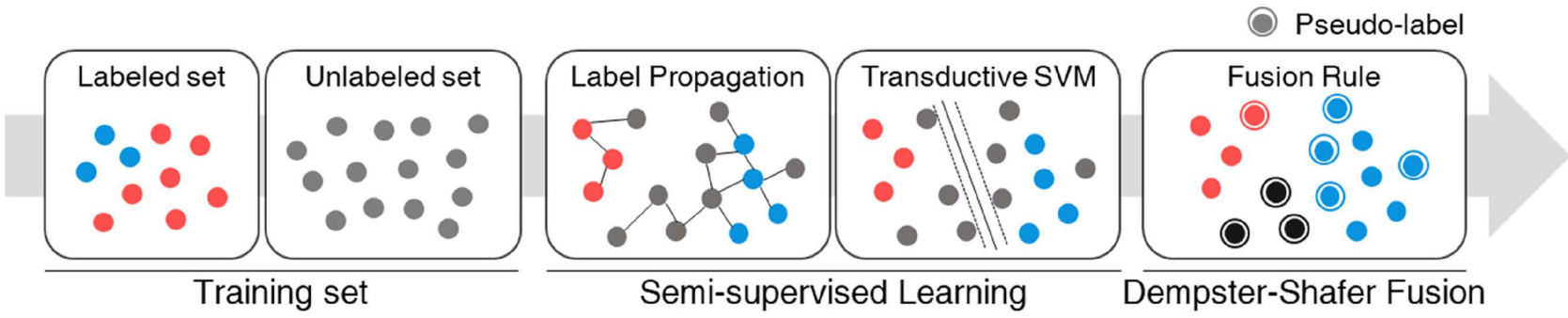
In [ ]:

```
fi = feature_importances_
fi_gain = fi[[col for col in fi.columns if col.startswith('gain')]].mean(axis=1)
fi_splt = fi[[col for col in fi.columns if col.startswith('split')]].mean(axis=1)

fig, ax = plt.subplots(1, 2, figsize=(14, 6), dpi=150)

# Split fi.
data_splt = fi_splt.sort_values(ascending=False)
sns.barplot(x=data_splt.values, y=data_splt.index,
            color='#1E90FF', linewidth=0.5, edgecolor="black", ax=ax[0])
ax[0].set_title(f'Feature Importance "Split"', fontdict={'fontweight': 'bold'})
ax[0].set_xlabel("Importance", fontsize=12)
ax[0].spines['right'].set_visible(False)
ax[0].spines['top'].set_visible(False)

# Gain fi.
data_gain = fi_splt.sort_values(ascending=False)
sns.barplot(x=data_gain.values, y=data_gain.index,
            color='#4169E1', linewidth=0.5, edgecolor="black", ax=ax[1])
ax[1].set_title(f'Feature Importance "Gain"', fontdict={'fontweight': 'bold'})
ax[1].set_xlabel("Importance", fontsize=12)
ax[1].spines['right'].set_visible(False)
ax[1].spines['top'].set_visible(False)

plt.tight_layout()
plt.show()

del eval_results_, models_, oof
```

Pseudo Labeling

In the following description, it is SVM, but any algorithm can be used. Here the pseudo labels are generated based on the Xgboost results. To avoid labeling uncertain data, only data with a high degree of confidence should be labeled. Here, class 0 is set to 0.999 and class 1 to 0.99.

Note: Pseudo-labels were tried on this data set, but did not lead to an increase in LB scores, so they are commented out.

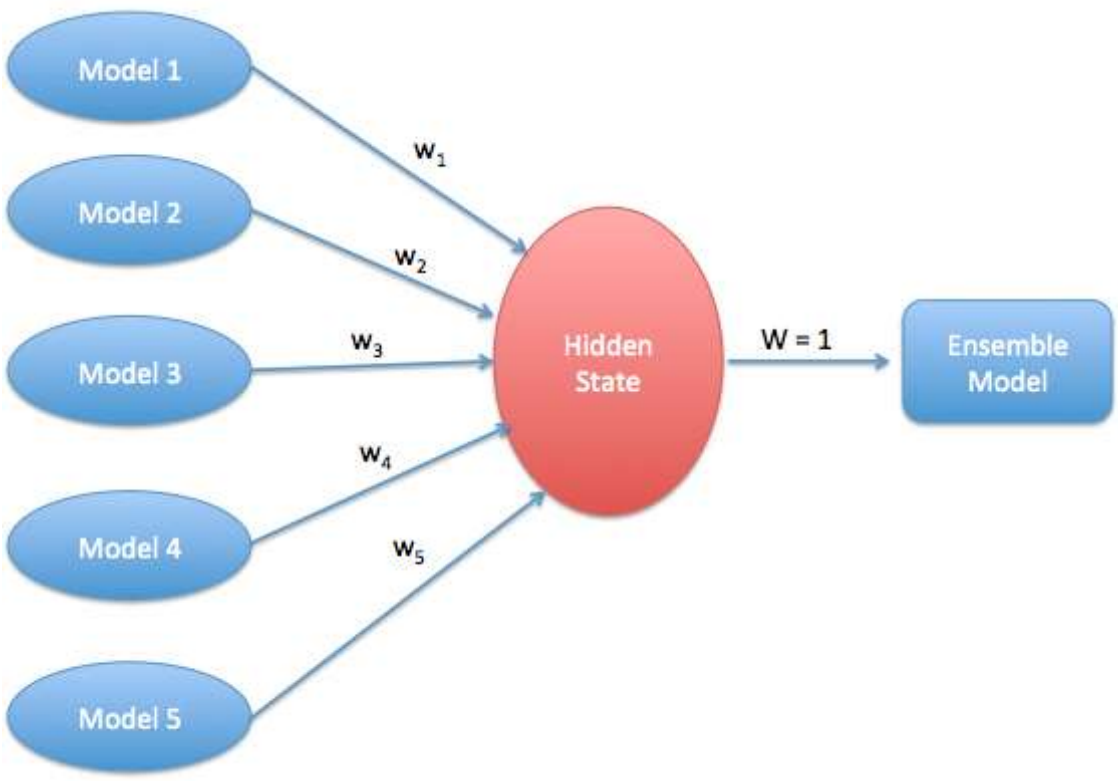


Reference. https://journals.sagepub.com/doi/10.1177/00368504221124004?icid=int.sj-abstract.similar-articles.4

In [ ]:
```
# df = pd.DataFrame(np.stack([1 - test_predss, test_predss], axis=1), columns=[f'{n}' for n in range(2)])
# true_idx, false_idx = df[df['1'] > true_th].index, df[df['0'] > false_th].index
# print(f'False label: {len(true_idx)}, ', f'True label: {len(false_idx)}')

# false_X_test = X_test.loc[false_idx].copy()
# false_X_test[target_col] = int(0)
# true_X_test = X_test.loc[true_idx].copy()
# true_X_test[target_col] = int(1)

# X_train = pd.concat([X_train_ori, true_X_test.drop(target_col, axis=1), false_X_test.drop(target_col, axis=1)], axis=0
# y_train = pd.concat([y_train_ori, true_X_test[target_col], false_X_test[target_col]], axis=0)

# X_train.reset_index(drop=True, inplace=True)
# y_train.reset_index(drop=True, inplace=True)

# print(f"X_train shape :{X_train.shape} , y_train shape :{y_train.shape}")
# print(f"X_test shape :{X_test.shape}")

# del df, true_idx, false_idx, true_X_test, false_X_test
```

Weighted Ensemble Model by Optuna on Training

A weighted average is performed during training;
The weights were determined for each model using the predictions for the train data created in the out of fold with Optuna's CMAsampler. (Here it is defined by `OptunaWeights`)
This is an extension of the averaging method. All models are assigned different weights defining the importance of each model for prediction.



Optimizer (--> Optimize AUC)

In [ ]:
```
class OptunaWeights:
    def __init__(self, random_state, n_trials=100):
        self.study = None
        self.weights = None
        self.random_state = random_state
        self.n_trials = n_trials

    def _objective(self, trial, y_true, y_preds):
        # Define the weights for the predictions from each model
        weights = [trial.suggest_float(f"weight{n}", 1e-15, 1) for n in range(len(y_preds))]

        # Calculate the weighted prediction
        weighted_pred = np.average(np.array(y_preds).T, axis=1, weights=weights)

        # Calculate the score for the weighted prediction
        score = metric(y_true, weighted_pred)
        return score

    def fit(self, y_true, y_preds):
        optuna.logging.set_verbosity(optuna.logging.ERROR)
        sampler = optuna.samplers.CmaEsSampler(seed=self.random_state)
```

```
        pruner = optuna.pruners.HyperbandPruner()
        self.study = optuna.create_study(sampler=sampler, pruner=pruner, study_name="OptunaWeights", direction='maximiz
        objective_partial = partial(self._objective, y_true=y_true, y_preds=y_preds)
        self.study.optimize(objective_partial, n_trials=self.n_trials)
        self.weights = [self.study.best_params[f"weight{n}"] for n in range(len(y_preds))]

    def predict(self, y_preds):
        assert self.weights is not None, 'OptunaWeights error, must be fitted before predict'
        weighted_pred = np.average(np.array(y_preds).T, axis=1, weights=self.weights)
        return weighted_pred

    def fit_predict(self, y_true, y_preds):
        self.fit(y_true, y_preds)
        return self.predict(y_preds)

    def weights(self):
        return self.weights
```

In [26]:

```
%%time

# Initialize an array for storing test predictions
classifier = Classifier(n_estimators, device, random_state)
test_predss = np.zeros((X_test.shape[0]))
oof_predss = np.zeros((X_train.shape[0], n_reapts))
ensemble_score, ensemble_score_ = [], []
weights = []
trained_models = dict(zip([_ for _ in classifier.models_name if ('xgb' in _) or ('lgb' in _) or ('cat' in _)], [[] for
score_dict = dict(zip(classifier.models_name, [[] for _ in range(classifier.len_models)]))

splitter = Splitter(kfold=kfold, n_splits=n_splits, cat_df=y_train)
for i, (X_train_, X_val, y_train_, y_val, val_index) in enumerate(splitter.split_data(X_train, y_train, random_state_li
n = i % n_splits
m = i // n_splits

# Get a set of classifier models
classifier = Classifier(n_estimators, device, random_state_list[m])
models = classifier.models

# Initialize lists to store oof and test predictions for each base model
oof_preds = []
test_preds = []

# Loop over each base model and fit it to the training data, evaluate on validation data, and store predictions
for name, model in models.items():
    best_iteration = None
    start_time = time.time()

    if ('xgb' in name) or ('lgb' in name) or ('cat' in name):
        early_stopping_rounds_ = int(early_stopping_rounds*2) if ('xgb' not in name) else early_stopping_rounds

        if 'lgb' in name:
            model.fit(
                X_train_, y_train_, eval_set=[(X_val, y_val)], categorical_feature=cat_cols,
                early_stopping_rounds=early_stopping_rounds_, verbose=verbose)
        elif 'cat' in name :
            model.fit(
                Pool(X_train_, y_train_, cat_features=cat_cols), eval_set=Pool(X_val, y_val, cat_features=cat_cols)
                early_stopping_rounds=early_stopping_rounds_, verbose=verbose)
        else:
            model.fit(X_train_, y_train_, eval_set=[(X_val, y_val)], early_stopping_rounds=early_stopping_rounds_,

        best_iteration = model.best_iteration if ('xgb' in name) else model.best_iteration_
    else:
        model.fit(X_train_, y_train_)

    end_time = time.time()
    min_, sec = sec_to_minsec(end_time - start_time)

    if name in trained_models.keys():
        trained_models[f'{name}'].append(deepcopy(model))

    y_val_pred = model.predict_proba(X_val)[:, 1].reshape(-1)
    test_pred = model.predict_proba(X_test)[:, 1].reshape(-1)

    score = metric(y_val, y_val_pred)
    score_dict[name].append(score)
    print(f'{blu}{name}{res} [FOLD-{n} SEED-{random_state_list[m]}] {metric_name} {blu}{score:.5f}{res} | Best iter

    oof_preds.append(y_val_pred)
    test_preds.append(test_pred)

# Use Optuna to find the best ensemble weights
optweights = OptunaWeights(random_state=random_state_list[m], n_trials=n_trials)
y_val_pred = optweights.fit_predict(y_val.values, oof_preds)

score = metric(y_val, y_val_pred)
print(f'{red}>>> Ensemble{res} [FOLD-{n} SEED-{random_state_list[m]}] {metric_name} {red}{score:.5f}{res}')
print(f'{"-" * 60}')
ensemble_score.append(score)
weights.append(optweights.weights)

# Predict to X_test by the best ensemble weights
test_predss += optweights.predict(test_preds) / (n_splits * len(random_state_list))
oof_predss[X_val.index, m] += optweights.predict(oof_preds)
```

```
    gc.collect()
```

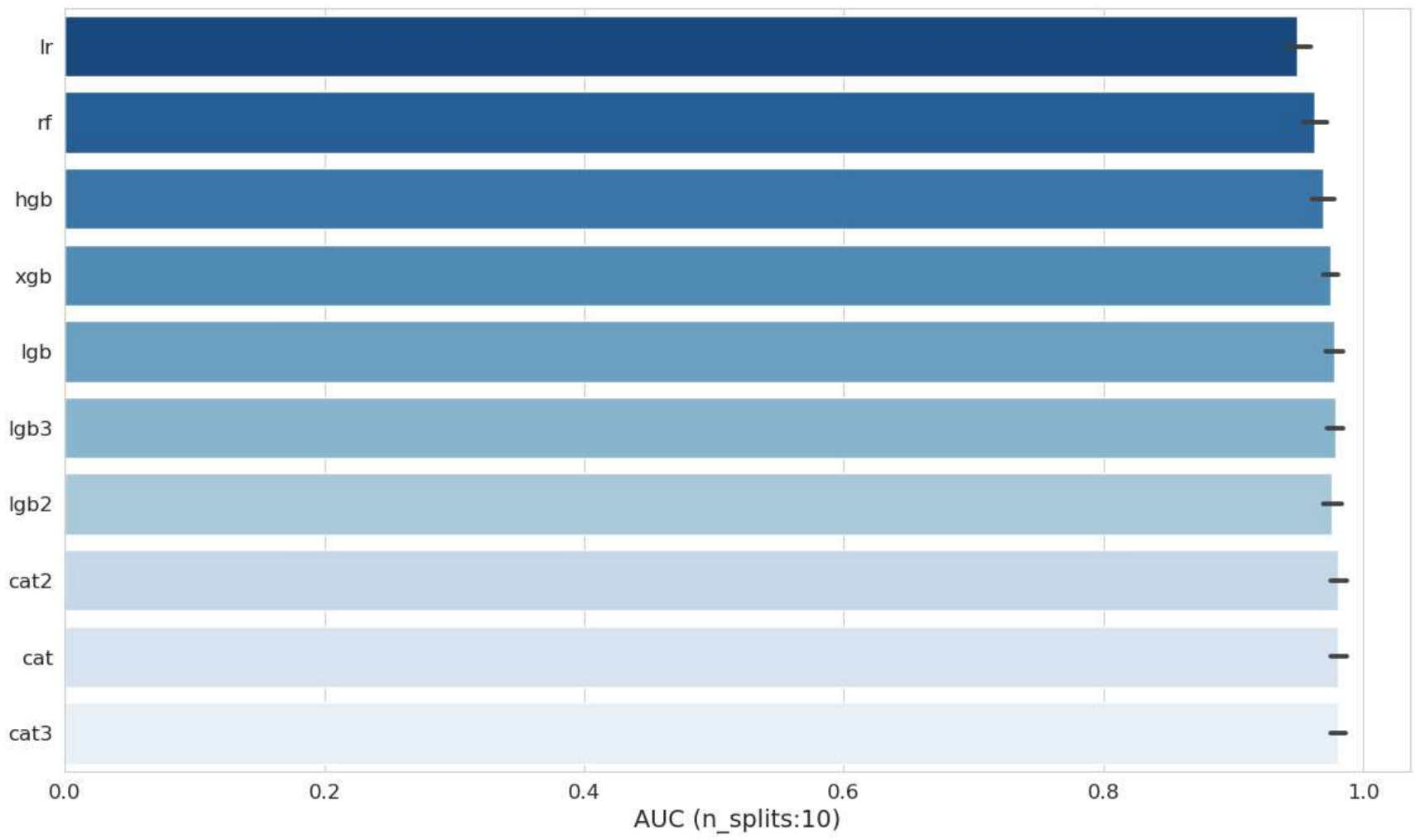
```
lgb3 [FOLD-1 SEED-1824] AUC 0.97068 | Best iteration 559 | Runtime 0min 26s
cat [FOLD-1 SEED-1824] AUC 0.97723 | Best iteration 833 | Runtime 0min 47s
cat2 [FOLD-1 SEED-1824] AUC 0.97739 | Best iteration 1230 | Runtime 1min 22s
cat3 [FOLD-1 SEED-1824] AUC 0.97816 | Best iteration 592 | Runtime 0min 51s
hgb [FOLD-1 SEED-1824] AUC 0.96967 | Best iteration None | Runtime 0min 31s
rf [FOLD-1 SEED-1824] AUC 0.96148 | Best iteration None | Runtime 1min 2s
lr [FOLD-1 SEED-1824] AUC 0.94644 | Best iteration None | Runtime 0min 22s
>>> Ensemble [FOLD-1 SEED-1824] AUC 0.97828
------------------------------------------------------------
xgb [FOLD-2 SEED-1824] AUC 0.96438 | Best iteration 561 | Runtime 0min 49s
lgb [FOLD-2 SEED-1824] AUC 0.96537 | Best iteration 1173 | Runtime 0min 42s
lgb2 [FOLD-2 SEED-1824] AUC 0.96714 | Best iteration 1227 | Runtime 0min 42s
lgb3 [FOLD-2 SEED-1824] AUC 0.96697 | Best iteration 199 | Runtime 0min 17s
cat [FOLD-2 SEED-1824] AUC 0.96879 | Best iteration 375 | Runtime 0min 30s
cat2 [FOLD-2 SEED-1824] AUC 0.96877 | Best iteration 813 | Runtime 1min 2s
cat3 [FOLD-2 SEED-1824] AUC 0.96895 | Best iteration 409 | Runtime 0min 41s
hgb [FOLD-2 SEED-1824] AUC 0.95246 | Best iteration None | Runtime 0min 25s
rf [FOLD-2 SEED-1824] AUC 0.94900 | Best iteration None | Runtime 1min 1s
lr [FOLD-2 SEED-1824] AUC 0.94384 | Best iteration None | Runtime 0min 27s
>>> Ensemble [FOLD-2 SEED-1824] AUC 0.97210
------------------------------------------------------------
xgb [FOLD-3 SEED-1824] AUC 0.97225 | Best iteration 652 | Runtime 0min 54s
lgb [FOLD-3 SEED-1824] AUC 0.97888 | Best iteration 754 | Runtime 0min 31s
lgb2 [FOLD-3 SEED-1824] AUC 0.98070 | Best iteration 492 | Runtime 0min 23s
lgb3 [FOLD-3 SEED-1824] AUC 0.98111 | Best iteration 606 | Runtime 0min 27s
cat [FOLD-3 SEED-1824] AUC 0.97899 | Best iteration 1125 | Runtime 0min 59s
cat2 [FOLD-3 SEED-1824] AUC 0.97870 | Best iteration 1459 | Runtime 1min 38s
cat3 [FOLD-3 SEED-1824] AUC 0.97983 | Best iteration 850 | Runtime 1min 8s
hgb [FOLD-3 SEED-1824] AUC 0.96667 | Best iteration None | Runtime 0min 33s
rf [FOLD-3 SEED-1824] AUC 0.96352 | Best iteration None | Runtime 1min 2s
lr [FOLD-3 SEED-1824] AUC 0.94953 | Best iteration None | Runtime 0min 22s
>>> Ensemble [FOLD-3 SEED-1824] AUC 0.98304
------------------------------------------------------------
xgb [FOLD-4 SEED-1824] AUC 0.97825 | Best iteration 575 | Runtime 0min 50s
lgb [FOLD-4 SEED-1824] AUC 0.98072 | Best iteration 1010 | Runtime 0min 38s
lgb2 [FOLD-4 SEED-1824] AUC 0.98058 | Best iteration 364 | Runtime 0min 20s
lgb3 [FOLD-4 SEED-1824] AUC 0.98081 | Best iteration 657 | Runtime 0min 29s
cat [FOLD-4 SEED-1824] AUC 0.98384 | Best iteration 958 | Runtime 0min 52s
cat2 [FOLD-4 SEED-1824] AUC 0.98194 | Best iteration 1593 | Runtime 1min 42s
cat3 [FOLD-4 SEED-1824] AUC 0.98221 | Best iteration 752 | Runtime 1min 0s
hgb [FOLD-4 SEED-1824] AUC 0.96679 | Best iteration None | Runtime 0min 31s
rf [FOLD-4 SEED-1824] AUC 0.97088 | Best iteration None | Runtime 1min 6s
lr [FOLD-4 SEED-1824] AUC 0.93862 | Best iteration None | Runtime 0min 25s
>>> Ensemble [FOLD-4 SEED-1824] AUC 0.98585
------------------------------------------------------------
xgb [FOLD-5 SEED-1824] AUC 0.98151 | Best iteration 684 | Runtime 0min 57s
lgb [FOLD-5 SEED-1824] AUC 0.98465 | Best iteration 625 | Runtime 0min 27s
lgb2 [FOLD-5 SEED-1824] AUC 0.98041 | Best iteration 430 | Runtime 0min 22s
lgb3 [FOLD-5 SEED-1824] AUC 0.98449 | Best iteration 795 | Runtime 0min 32s
cat [FOLD-5 SEED-1824] AUC 0.98995 | Best iteration 1254 | Runtime 1min 4s
cat2 [FOLD-5 SEED-1824] AUC 0.98803 | Best iteration 1629 | Runtime 1min 44s
cat3 [FOLD-5 SEED-1824] AUC 0.98772 | Best iteration 945 | Runtime 1min 11s
hgb [FOLD-5 SEED-1824] AUC 0.97974 | Best iteration None | Runtime 0min 32s
rf [FOLD-5 SEED-1824] AUC 0.97722 | Best iteration None | Runtime 1min 4s
lr [FOLD-5 SEED-1824] AUC 0.96696 | Best iteration None | Runtime 0min 20s
>>> Ensemble [FOLD-5 SEED-1824] AUC 0.99064
------------------------------------------------------------
xgb [FOLD-6 SEED-1824] AUC 0.97779 | Best iteration 558 | Runtime 0min 49s
lgb [FOLD-6 SEED-1824] AUC 0.97973 | Best iteration 218 | Runtime 0min 16s
lgb2 [FOLD-6 SEED-1824] AUC 0.97863 | Best iteration 876 | Runtime 0min 33s
lgb3 [FOLD-6 SEED-1824] AUC 0.98041 | Best iteration 235 | Runtime 0min 17s
cat [FOLD-6 SEED-1824] AUC 0.98320 | Best iteration 1263 | Runtime 1min 3s
cat2 [FOLD-6 SEED-1824] AUC 0.98486 | Best iteration 1403 | Runtime 1min 32s
cat3 [FOLD-6 SEED-1824] AUC 0.98417 | Best iteration 930 | Runtime 1min 8s
hgb [FOLD-6 SEED-1824] AUC 0.96874 | Best iteration None | Runtime 0min 36s
rf [FOLD-6 SEED-1824] AUC 0.95772 | Best iteration None | Runtime 1min 3s
lr [FOLD-6 SEED-1824] AUC 0.94278 | Best iteration None | Runtime 0min 20s
>>> Ensemble [FOLD-6 SEED-1824] AUC 0.98506
------------------------------------------------------------
xgb [FOLD-7 SEED-1824] AUC 0.97913 | Best iteration 663 | Runtime 0min 56s
lgb [FOLD-7 SEED-1824] AUC 0.97828 | Best iteration 731 | Runtime 0min 30s
lgb2 [FOLD-7 SEED-1824] AUC 0.97868 | Best iteration 553 | Runtime 0min 24s
lgb3 [FOLD-7 SEED-1824] AUC 0.97917 | Best iteration 772 | Runtime 0min 31s
cat [FOLD-7 SEED-1824] AUC 0.98210 | Best iteration 622 | Runtime 0min 39s
cat2 [FOLD-7 SEED-1824] AUC 0.98278 | Best iteration 1204 | Runtime 1min 21s
cat3 [FOLD-7 SEED-1824] AUC 0.98163 | Best iteration 667 | Runtime 0min 54s
hgb [FOLD-7 SEED-1824] AUC 0.97848 | Best iteration None | Runtime 0min 35s
rf [FOLD-7 SEED-1824] AUC 0.96363 | Best iteration None | Runtime 1min 2s
lr [FOLD-7 SEED-1824] AUC 0.95462 | Best iteration None | Runtime 0min 23s
>>> Ensemble [FOLD-7 SEED-1824] AUC 0.98473
------------------------------------------------------------
xgb [FOLD-8 SEED-1824] AUC 0.97318 | Best iteration 464 | Runtime 0min 43s
lgb [FOLD-8 SEED-1824] AUC 0.97785 | Best iteration 425 | Runtime 0min 22s
lgb2 [FOLD-8 SEED-1824] AUC 0.97074 | Best iteration 638 | Runtime 0min 28s
lgb3 [FOLD-8 SEED-1824] AUC 0.98061 | Best iteration 138 | Runtime 0min 14s
cat [FOLD-8 SEED-1824] AUC 0.97785 | Best iteration 614 | Runtime 0min 40s
cat2 [FOLD-8 SEED-1824] AUC 0.97779 | Best iteration 861 | Runtime 1min 4s
cat3 [FOLD-8 SEED-1824] AUC 0.97835 | Best iteration 379 | Runtime 0min 40s
hgb [FOLD-8 SEED-1824] AUC 0.96351 | Best iteration None | Runtime 0min 31s
rf [FOLD-8 SEED-1824] AUC 0.95252 | Best iteration None | Runtime 1min 2s
lr [FOLD-8 SEED-1824] AUC 0.93673 | Best iteration None | Runtime 0min 28s
>>> Ensemble [FOLD-8 SEED-1824] AUC 0.98257
------------------------------------------------------------
xgb [FOLD-9 SEED-1824] AUC 0.96957 | Best iteration 598 | Runtime 0min 52s
lgb [FOLD-9 SEED-1824] AUC 0.97301 | Best iteration 1100 | Runtime 0min 40s
```

```
lgb2 [FOLD-9 SEED-1824] AUC 0.96818 | Best iteration 1271 | Runtime 0min 44s
lgb3 [FOLD-9 SEED-1824] AUC 0.97471 | Best iteration 350 | Runtime 0min 19s
cat [FOLD-9 SEED-1824] AUC 0.98085 | Best iteration 353 | Runtime 0min 29s
cat2 [FOLD-9 SEED-1824] AUC 0.98143 | Best iteration 769 | Runtime 0min 59s
cat3 [FOLD-9 SEED-1824] AUC 0.98073 | Best iteration 348 | Runtime 0min 38s
hgb [FOLD-9 SEED-1824] AUC 0.96484 | Best iteration None | Runtime 0min 33s
rf [FOLD-9 SEED-1824] AUC 0.96030 | Best iteration None | Runtime 1min 0s
lr [FOLD-9 SEED-1824] AUC 0.95134 | Best iteration None | Runtime 0min 21s
>>> Ensemble [FOLD-9 SEED-1824] AUC 0.98192
------------------------------------------------------------
CPU times: user 4h 27min 31s, sys: 18min 4s, total: 4h 45min 36s
Wall time: 1h 28min 49s
```

Mean Scores for each model

In [27]:
```python
def plot_score_from_dict(score_dict, title='', ascending=True):
    score_df = pd.melt(pd.DataFrame(score_dict))
    score_df = score_df.sort_values('value', ascending=ascending)

    plt.figure(figsize=(14, 8))
    sns.barplot(x='value', y='variable', data=score_df, palette='Blues_r', errorbar='sd')
    plt.xlabel(f'{title}', fontsize=14)
    plt.ylabel('')
    #plt.title(f'{title}', fontsize=18)
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
    plt.grid(True, axis='x')
    plt.show()

print(f'--- Mean {metric_name} Scores---')
for name, score in score_dict.items():
    mean_score = np.mean(score)
    std_score = np.std(score)
    print(f'{name}: {red}{mean_score:.5f} ± {std_score:.5f}{res}')
plot_score_from_dict(score_dict, title=f'{metric_name} (n_splits:{n_splits})')
```

```
--- Mean AUC Scores---
xgb: 0.97532 ± 0.00564
lgb: 0.97763 ± 0.00633
lgb2: 0.97647 ± 0.00658
lgb3: 0.97868 ± 0.00593
cat: 0.98118 ± 0.00578
cat2: 0.98120 ± 0.00576
cat3: 0.98115 ± 0.00544
hgb: 0.96942 ± 0.00860
rf: 0.96314 ± 0.00868
lr: 0.94956 ± 0.00966
```

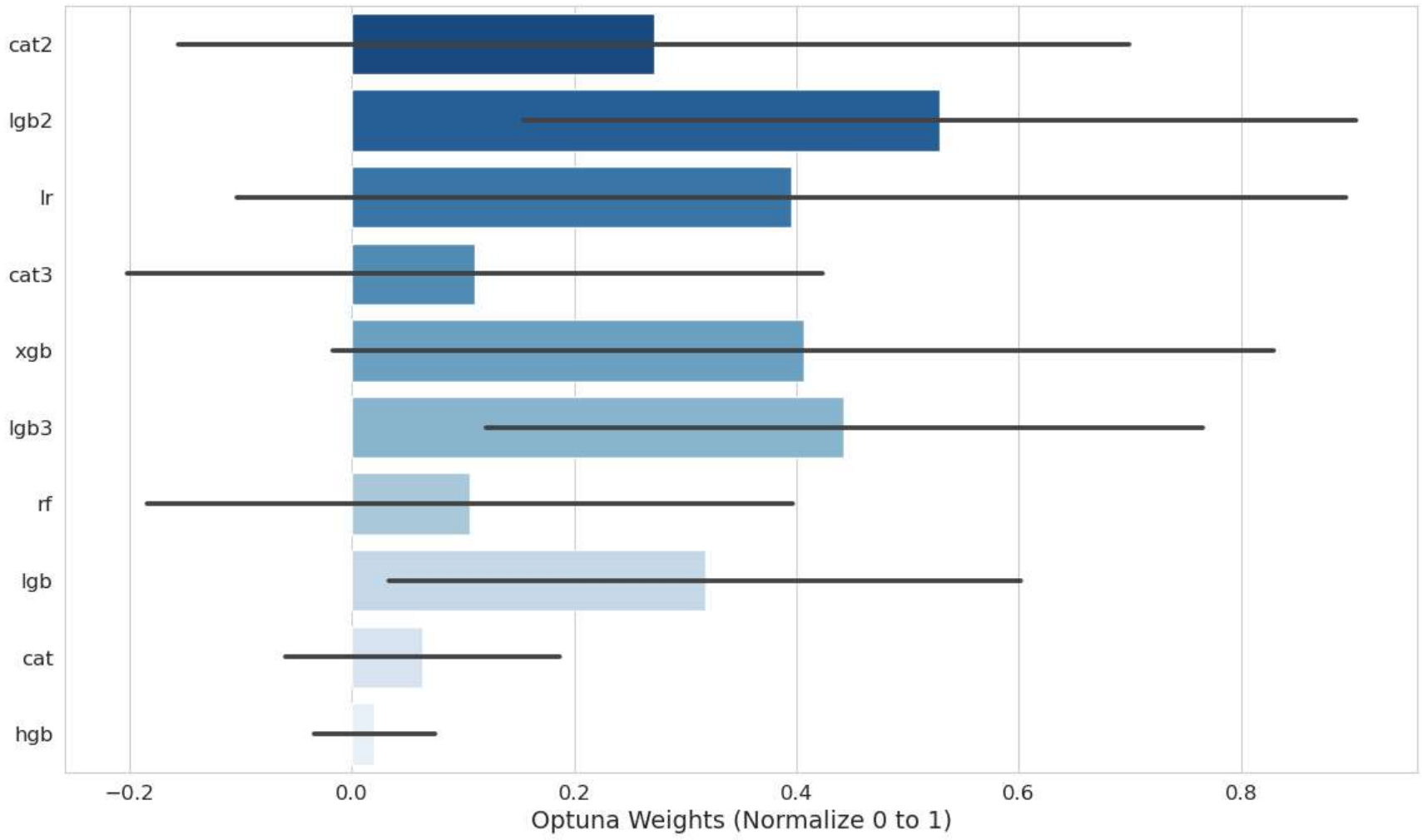


Weight of the Optuna Ensemble

In [28]:
```python
# Calculate the mean LogLoss score of the ensemble
mean_score = np.mean(ensemble_score)
std_score = np.std(ensemble_score)
print(f'{red}Mean{res} Optuna Ensemble {metric_name} {red}{mean_score:.5f} ± {std_score:.5f}{res}')

print('')
# Print the mean and standard deviation of the ensemble weights for each model
print('--- Optuna Weights---')
mean_weights = np.mean(weights, axis=0)
std_weights = np.std(weights, axis=0)
for name, mean_weight, std_weight in zip(models.keys(), mean_weights, std_weights):
    print(f'{name}: {blu}{mean_weight:.5f} ± {std_weight:.5f}{res}')
```

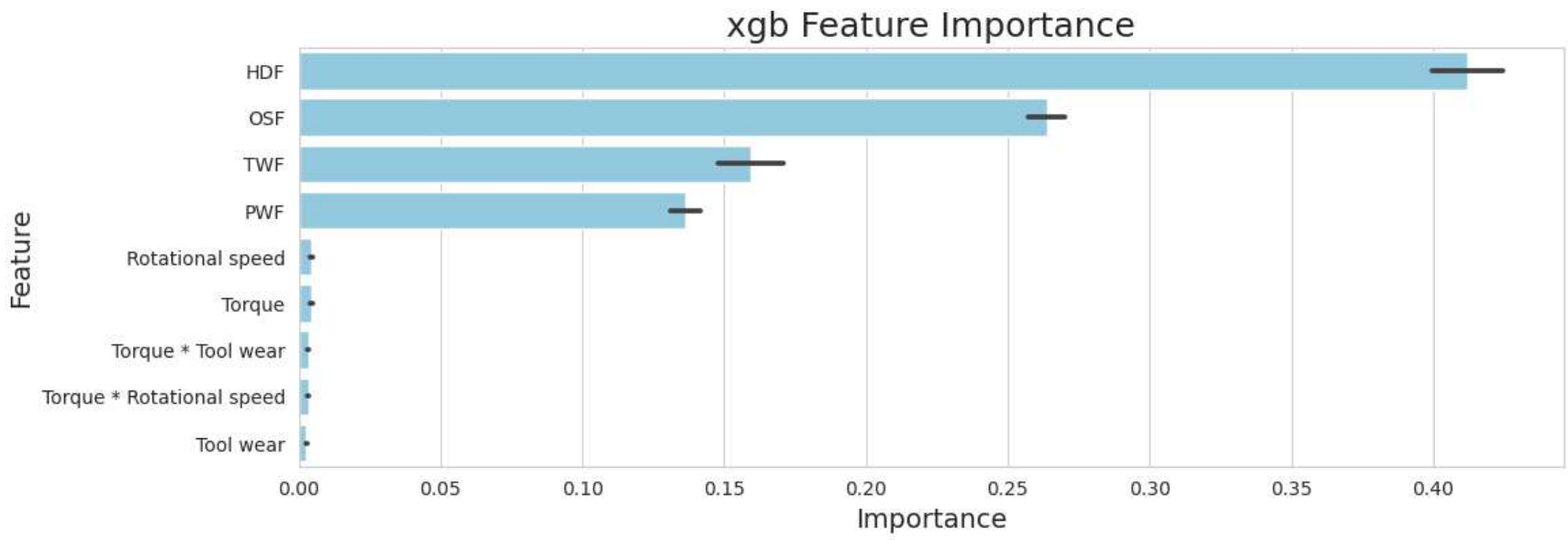
```
# weight_dict = dict(zip(list(score_dict.keys()), np.array(weights).T.tolist()))
# plot_score_from_dict(weight_dict, title='Model Weights', ascending=False)
normalize = [((weight - np.min(weight)) / (np.max(weight) - np.min(weight))).tolist() for weight in weights]
weight_dict = dict(zip(list(score_dict.keys()), np.array(normalize).T.tolist()))
plot_score_from_dict(weight_dict, title='Optuna Weights (Normalize 0 to 1)', ascending=False)
```

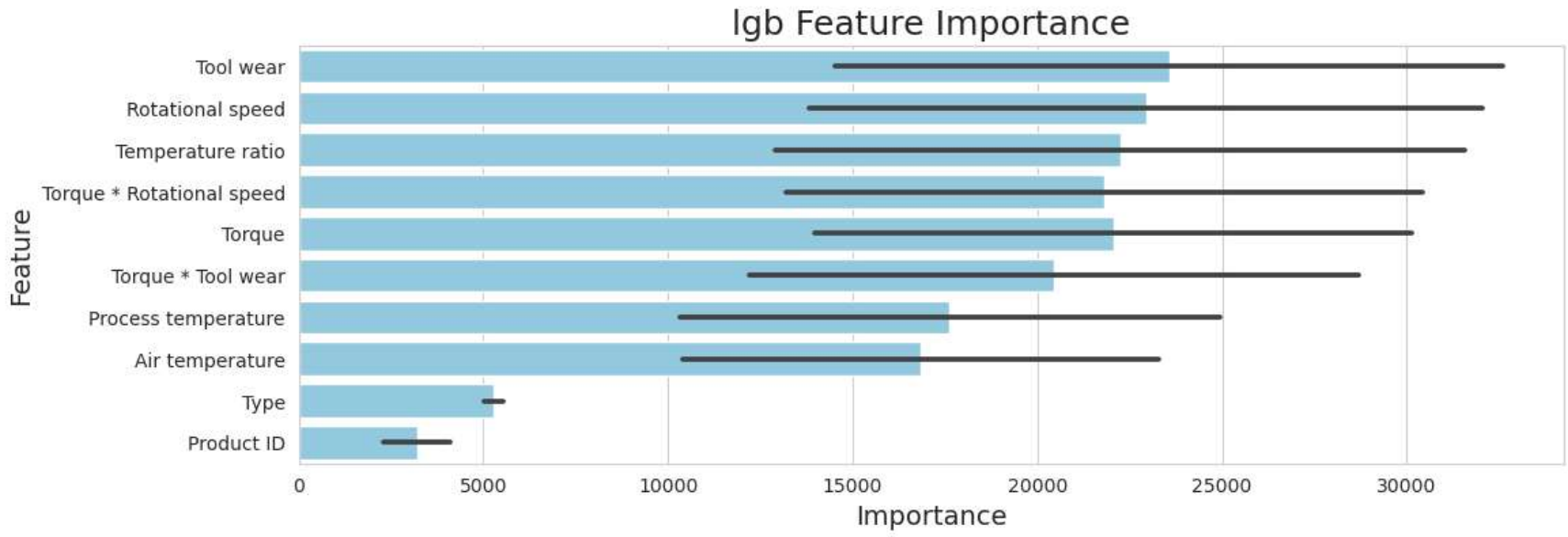
```
Mean Optuna Ensemble AUC 0.98346 ± 0.00518

--- Optuna Weights---
xgb: 0.38295 ± 0.38118
lgb: 0.28717 ± 0.24469
lgb2: 0.47932 ± 0.31565
lgb3: 0.40134 ± 0.26701
cat: 0.05855 ± 0.10332
cat2: 0.26022 ± 0.38714
cat3: 0.10240 ± 0.27616
hgb: 0.01977 ± 0.04993
rf: 0.10373 ± 0.26883
lr: 0.37216 ± 0.44611
```

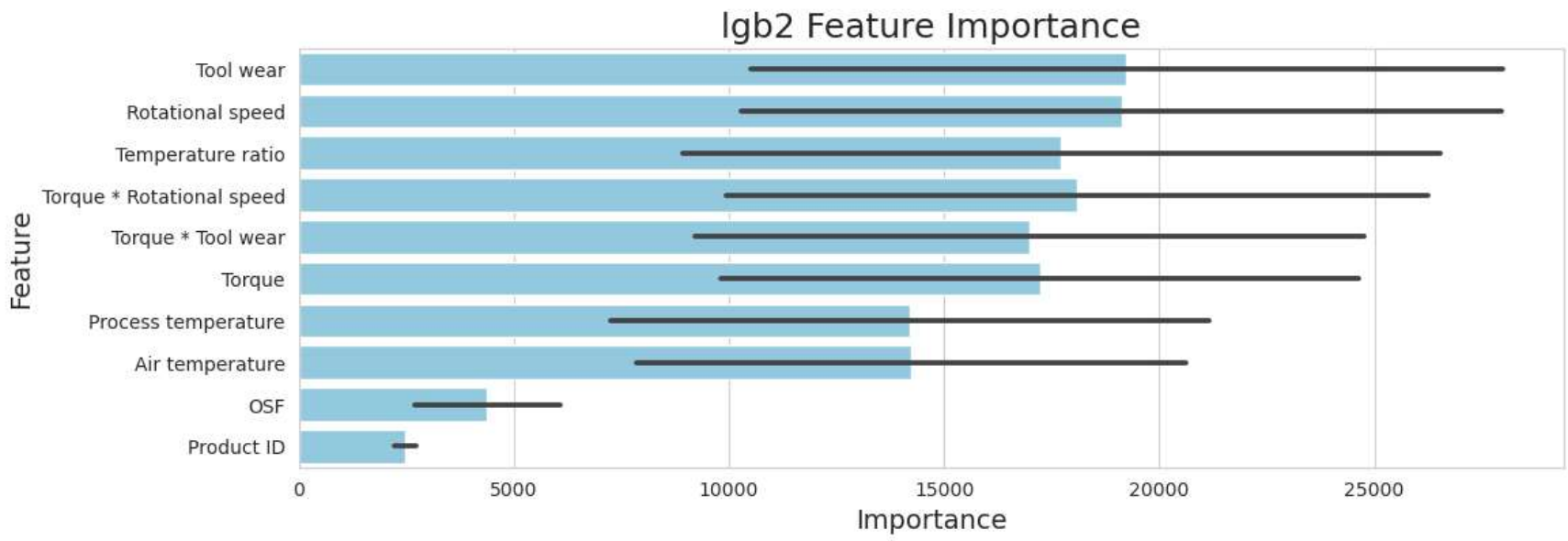


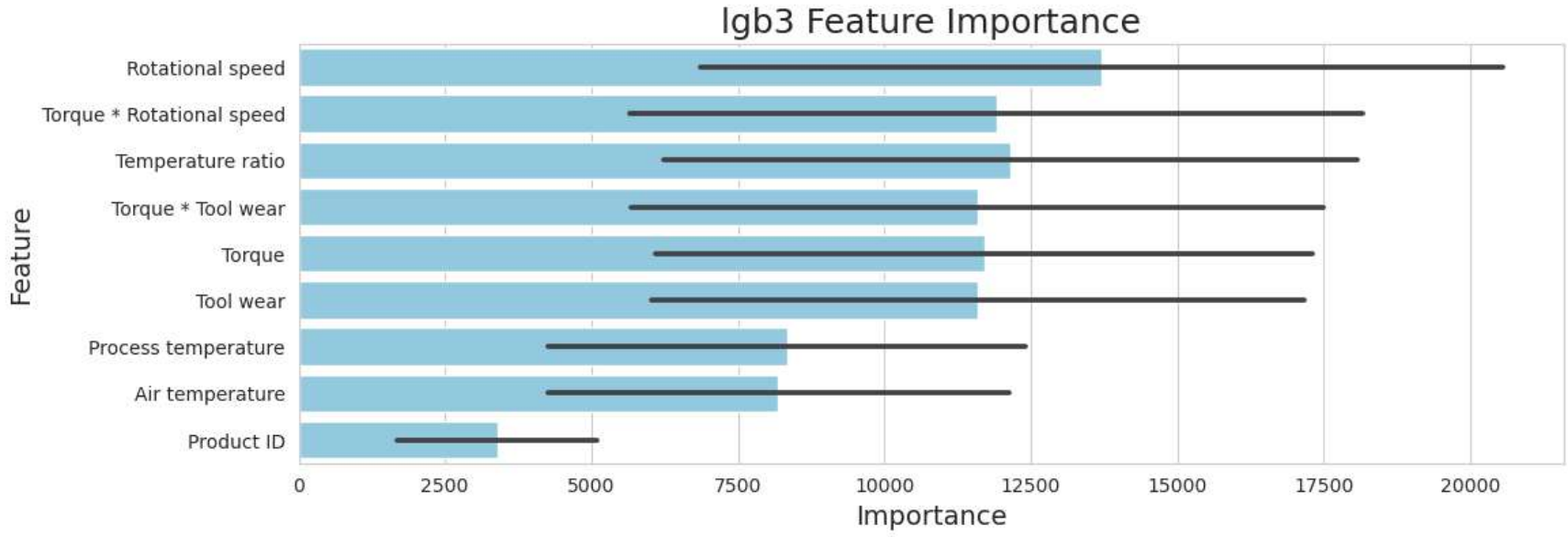
Feature importance Visualization

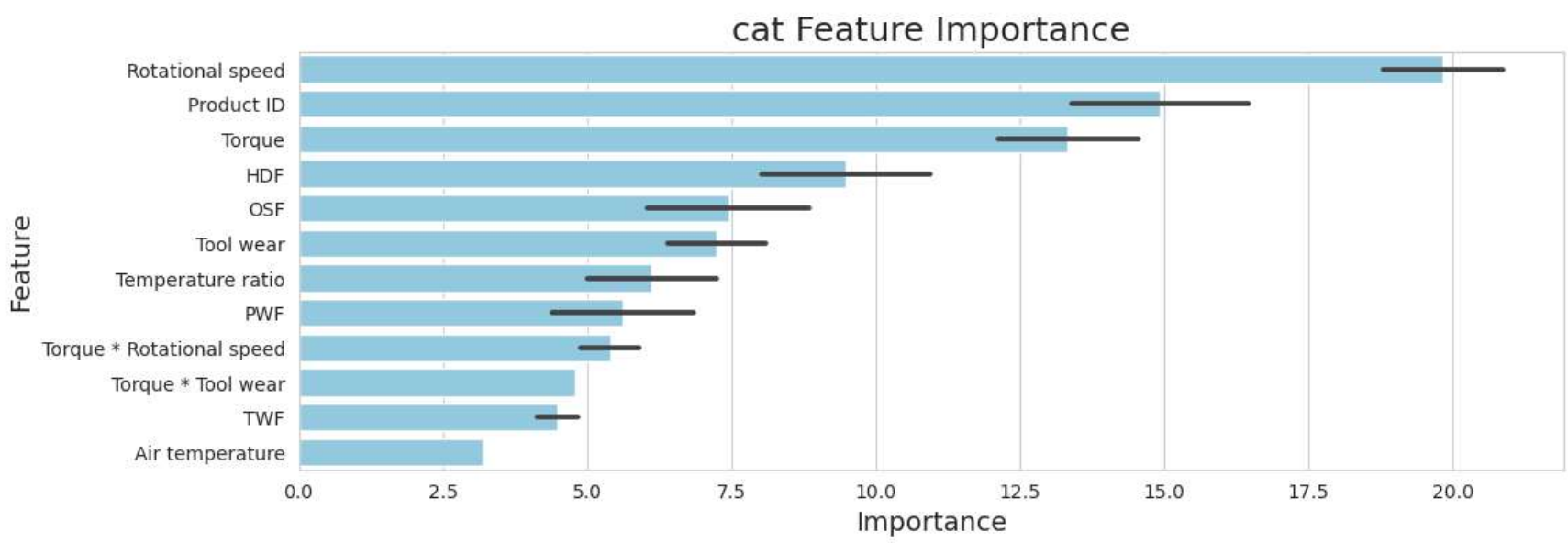
In [29]:
```
def visualize_importance(models, feature_cols, title, top=9):
    importances = []
    feature_importance = pd.DataFrame()
    for i, model in enumerate(models):
        _df = pd.DataFrame()
        _df["importance"] = model.feature_importances_
        _df["feature"] = pd.Series(feature_cols)
        _df["fold"] = i
        _df = _df.sort_values('importance', ascending=False)
        _df = _df.head(top)
        feature_importance = pd.concat([feature_importance, _df], axis=0, ignore_index=True)

    feature_importance = feature_importance.sort_values('importance', ascending=False)
    # display(feature_importance.groupby(["feature"]).mean().reset_index().drop('fold', axis=1))
    plt.figure(figsize=(12, 4))
    sns.barplot(x='importance', y='feature', data=feature_importance, color='skyblue', errorbar='sd')
    plt.xlabel('Importance', fontsize=14)
    plt.ylabel('Feature', fontsize=14)
    plt.title(f'{title} Feature Importance', fontsize=18)
    plt.grid(True, axis='x')
    plt.show()

for name, models in trained_models.items():
    if name in list(trained_models.keys()):
        visualize_importance(models, list(X_train.columns), name)
```

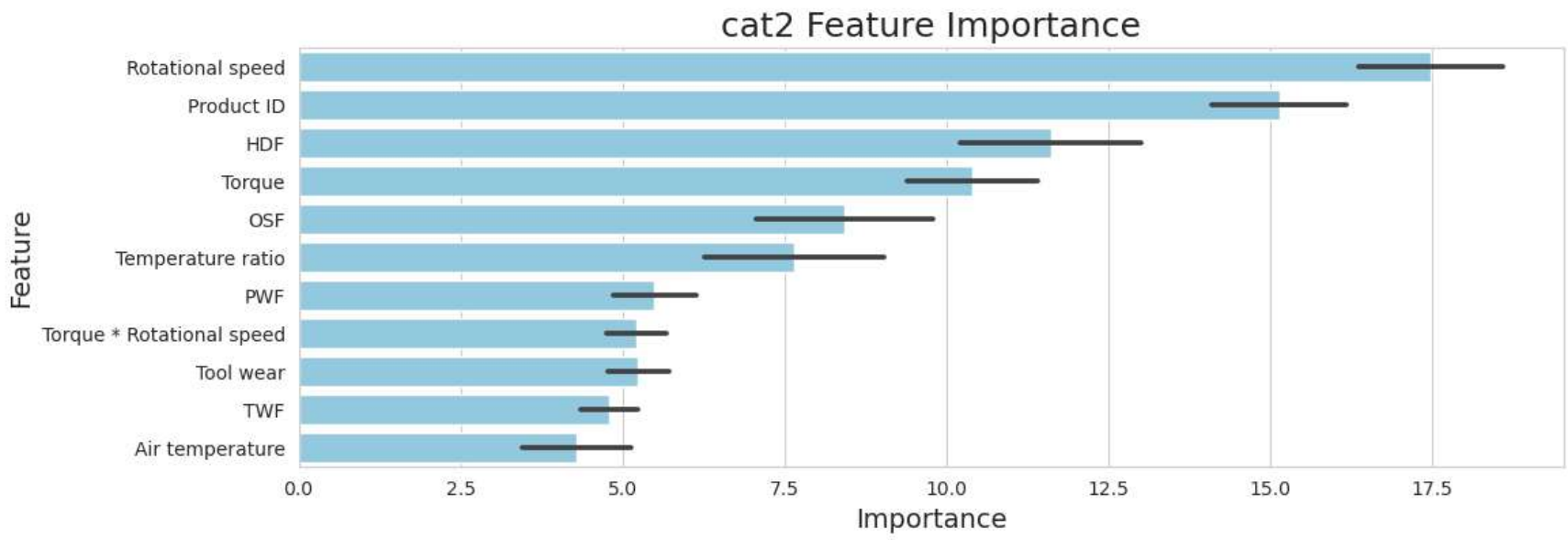

xgb Feature Importance
HDF
OSF
TWF
PWF
Rotational speed
Torque
Torque * Tool wear
Torque * Rotational speed
Tool wear
Feature
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Importance

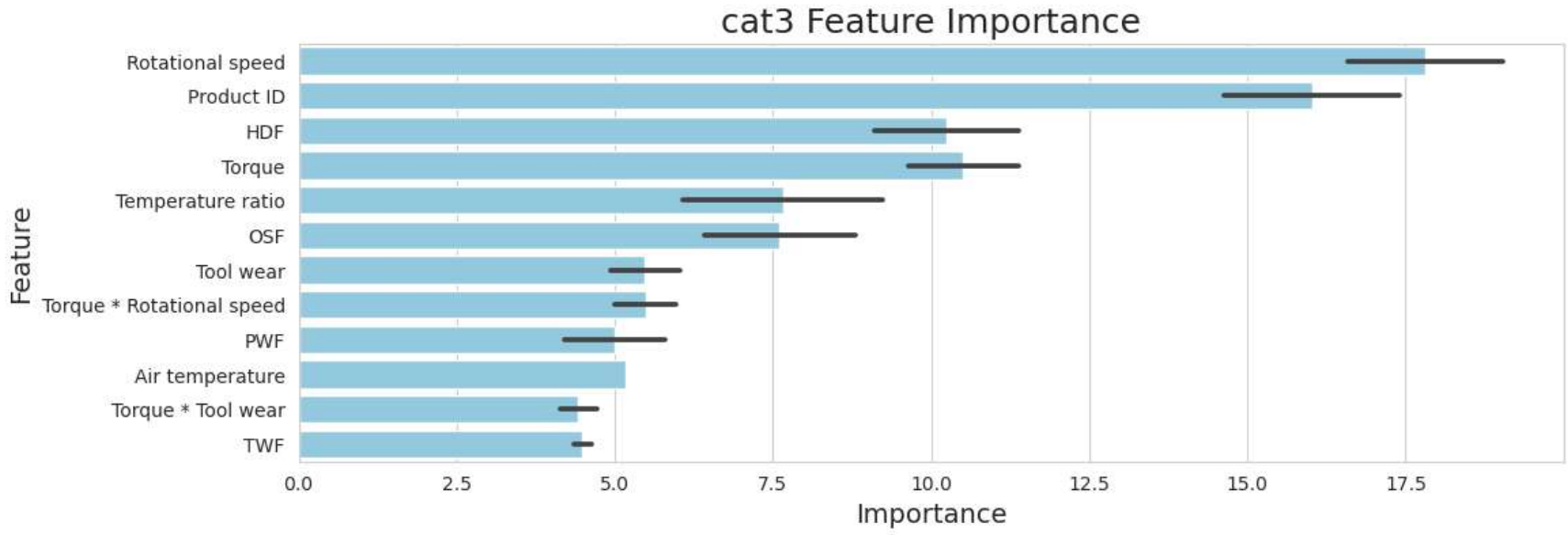

lgb Feature Importance
Tool wear
Rotational speed
Temperature ratio
Torque * Rotational speed
Torque
Torque * Tool wear
Process temperature
Air temperature
OSF
Type
Product ID
Feature
0
5000
10000
15000
20000
25000
30000
Importance


lgb2 Feature Importance
Tool wear
Rotational speed
Temperature ratio
Torque * Rotational speed
Torque * Tool wear
Torque
Process temperature
Air temperature
OSF
Product ID
Feature
0
5000
10000
15000
20000
25000
Importance


lgb3 Feature Importance
Rotational speed
Torque * Rotational speed
Temperature ratio
Torque * Tool wear
Torque
Tool wear
Process temperature
Air temperature
Product ID
Feature
0
2500
5000
7500
10000
12500
15000
17500
20000
Importance


cat Feature Importance
Rotational speed
Product ID
Torque
HDF
OSF
Tool wear
Temperature ratio
PWF
Torque * Rotational speed
Torque * Tool wear
TWF
Air temperature
Feature
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
Importance

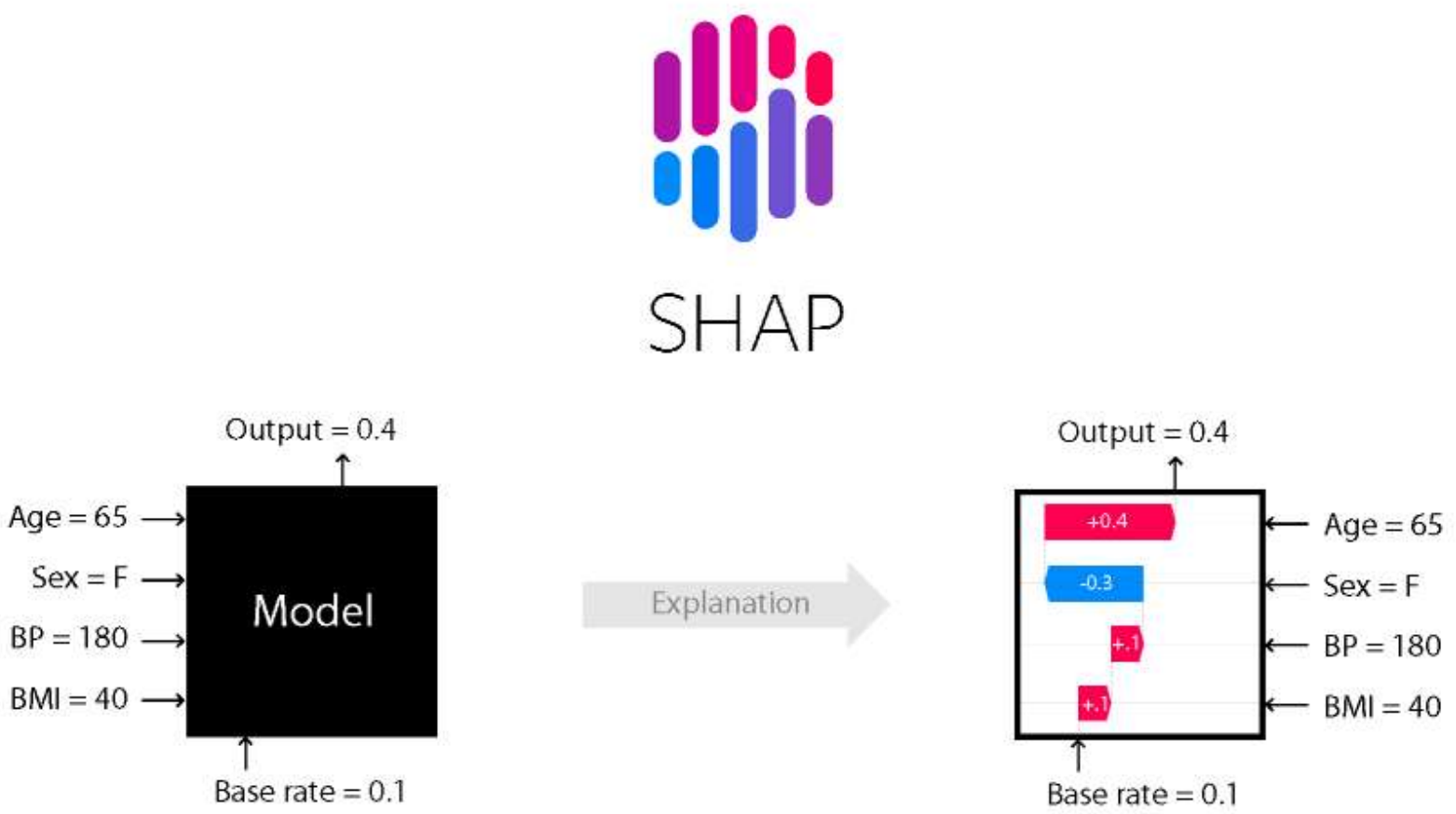




SHAP Analysis

SHAP stands for SHapley Additive exPlanations, a method for determining the contribution of each variable (feature) to the model's predicted outcome. Since SHAP cannot be adapted for ensemble models, let's use SHAP to understand `Xgboost` and `Catboost`.

Consideration of Results:
The `HDF` appears to be the most effective, as it comes the highest in both Catboost and Xgboost. This feature is binary, though. Next in importance is `Rotational speed`. This is understandably tied to `Machine failure`.



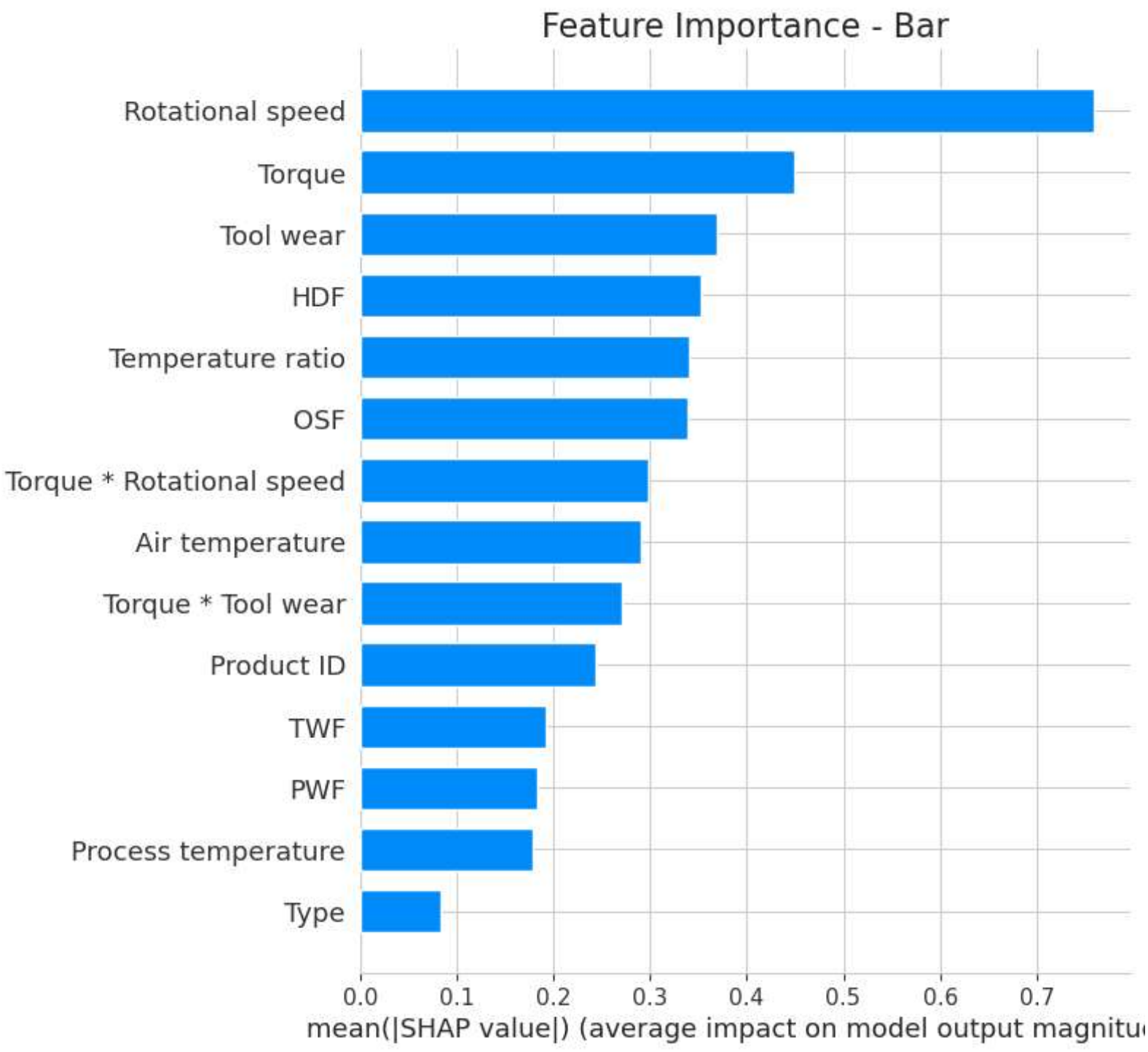
Reference1. https://meichenlu.com/2018-11-10-SHAP-explainable-machine-learning/
Reference2. https://christophm.github.io/interpretable-ml-book/shap.html

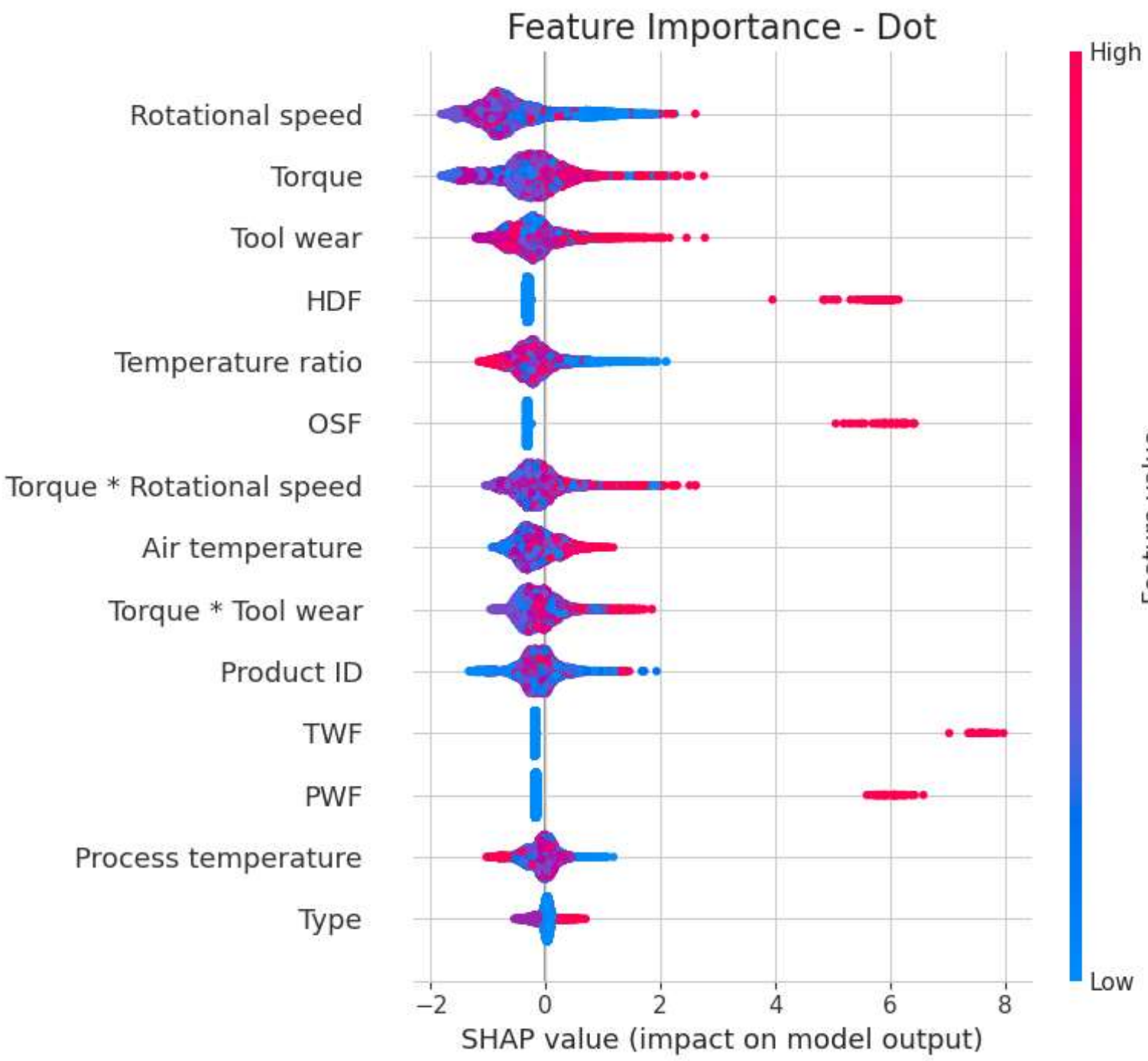
Xgboost

In [30]:
```
shap.initjs()
explainer = shap.TreeExplainer(model=trained_models['xgb'][-1])
shap_values = explainer.shap_values(X=X_val)
```

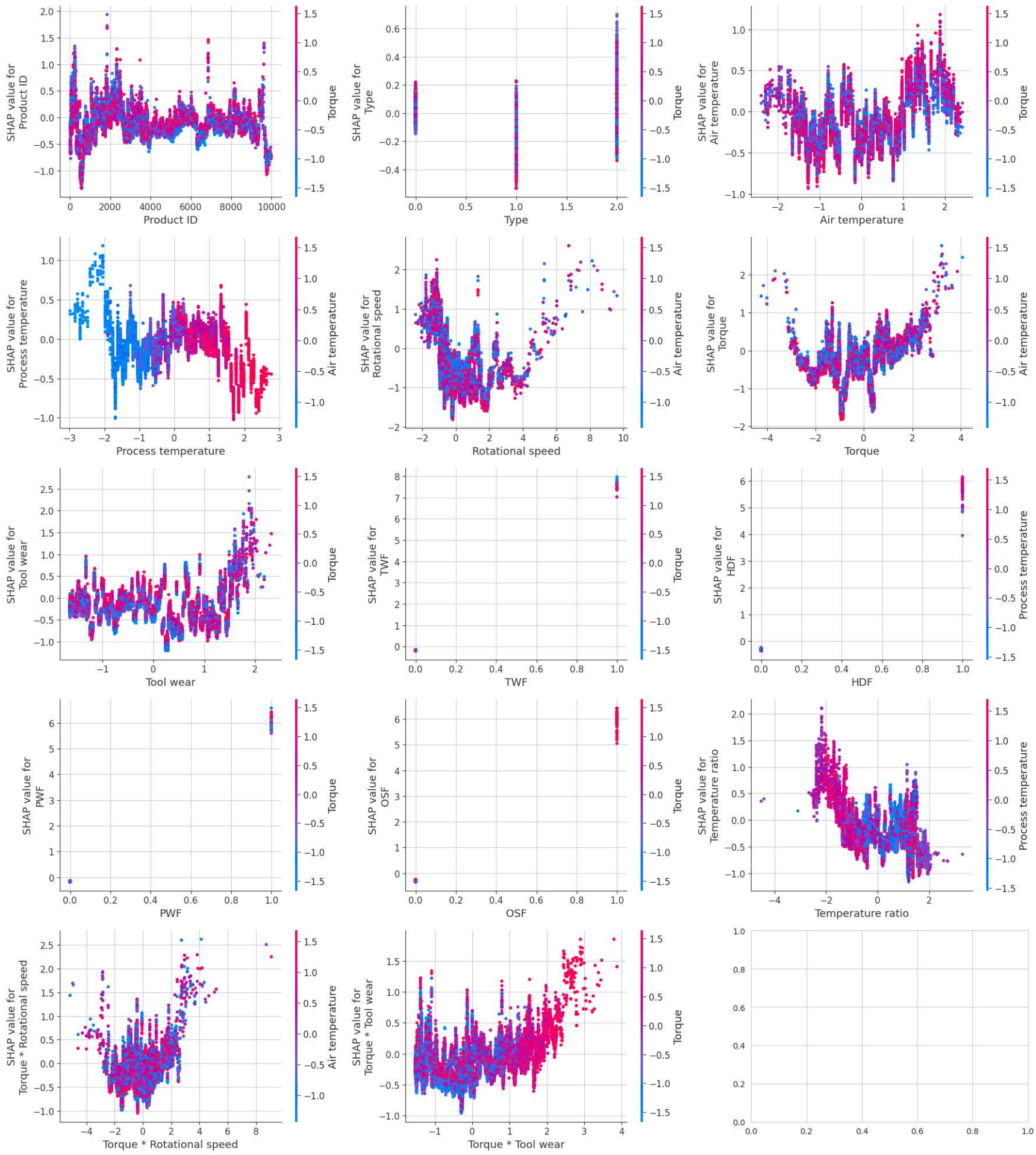
In [31]:
```
# Bar plot
plt.figure(figsize=(20, 14))
```

```
shap.summary_plot(shap_values, X_val, plot_type="bar", show=False)
plt.title("Feature Importance - Bar", fontsize=16)

# Dot plot
plt.figure(figsize=(20, 14))
shap.summary_plot(shap_values, X_val, plot_type="dot", show=False)
plt.title("Feature Importance - Dot", fontsize=16)

# Adjust layout and display the plots side by side
plt.tight_layout()
plt.show()
```





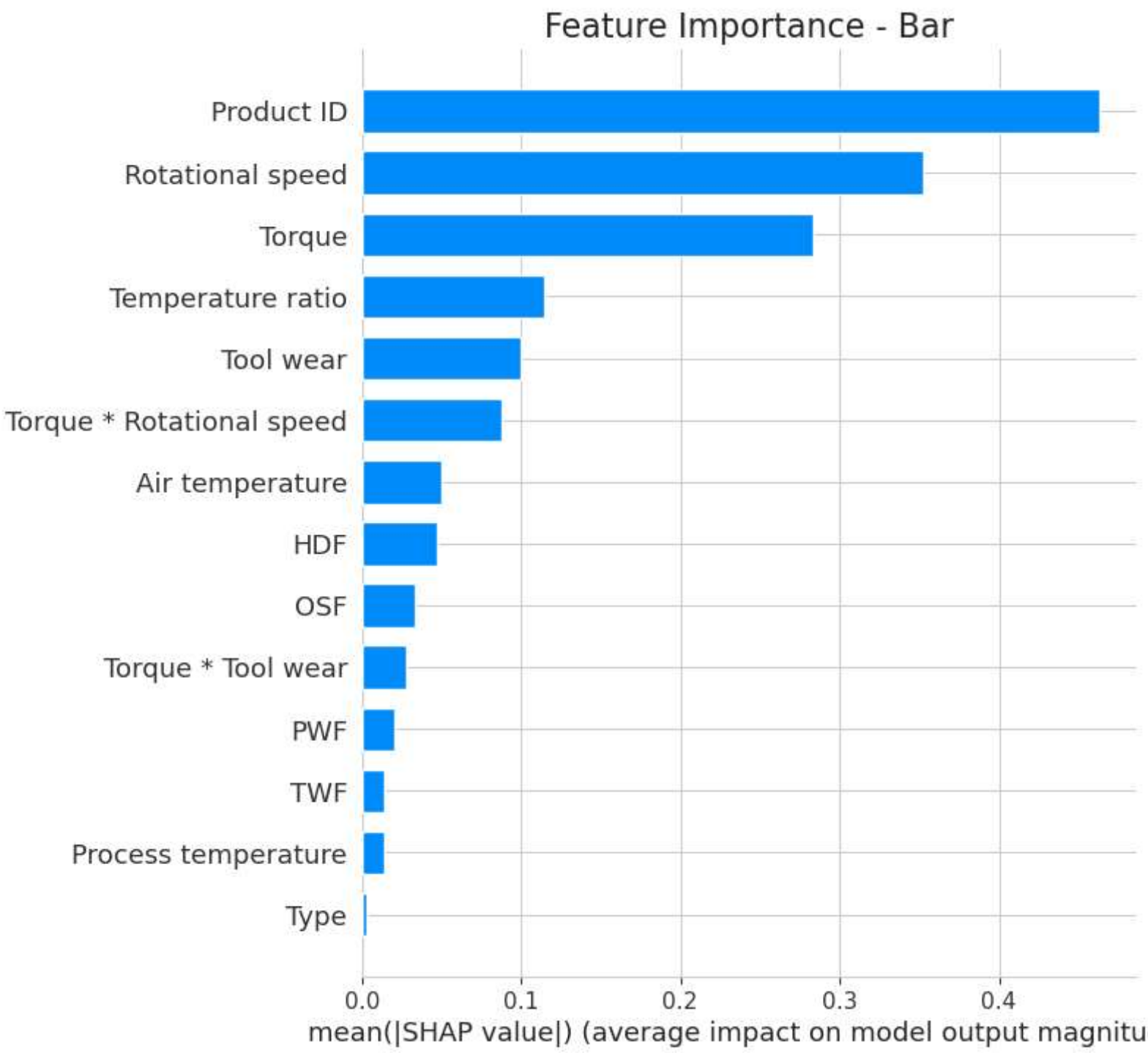
In [32]:
```
num_cols = min(3, len(X_val.columns))
num_rows = (len(X_val.columns) + num_cols - 1) // num_cols

fig, axes = plt.subplots(nrows=num_rows, ncols=num_cols, figsize=(6*num_cols, 4*num_rows))

for i, ind in enumerate(X_val.columns):
    row = i // num_cols
    col = i % num_cols

    shap.dependence_plot(ind=ind, shap_values=shap_values, features=X_val, feature_names=X_val.columns, ax=axes[row, co

plt.tight_layout()
plt.show()
```

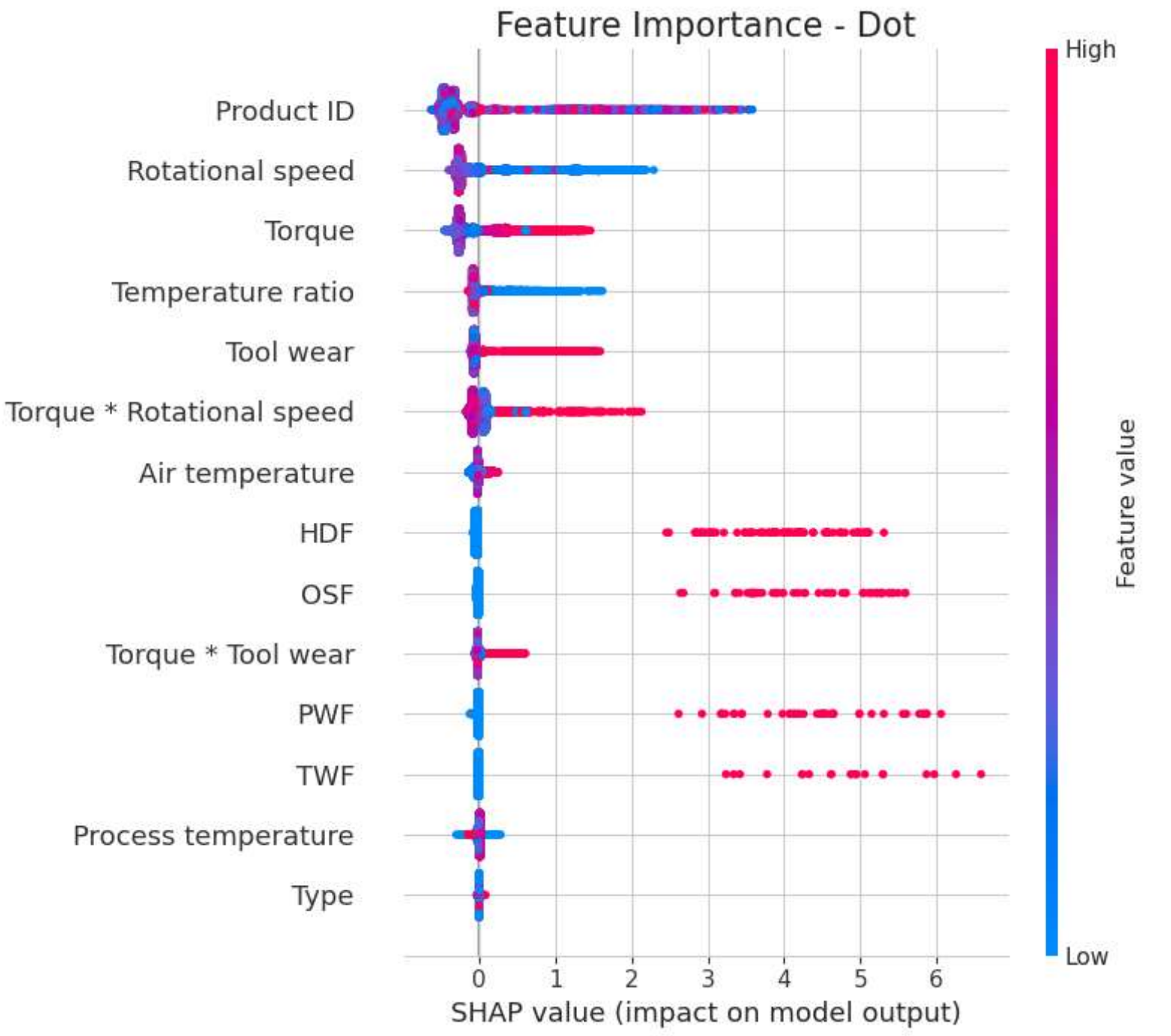


Catboost

In [33]:
```
explainer = shap.TreeExplainer(model=trained_models['cat'][-1])
shap_values = explainer.shap_values(X=X_val)
```

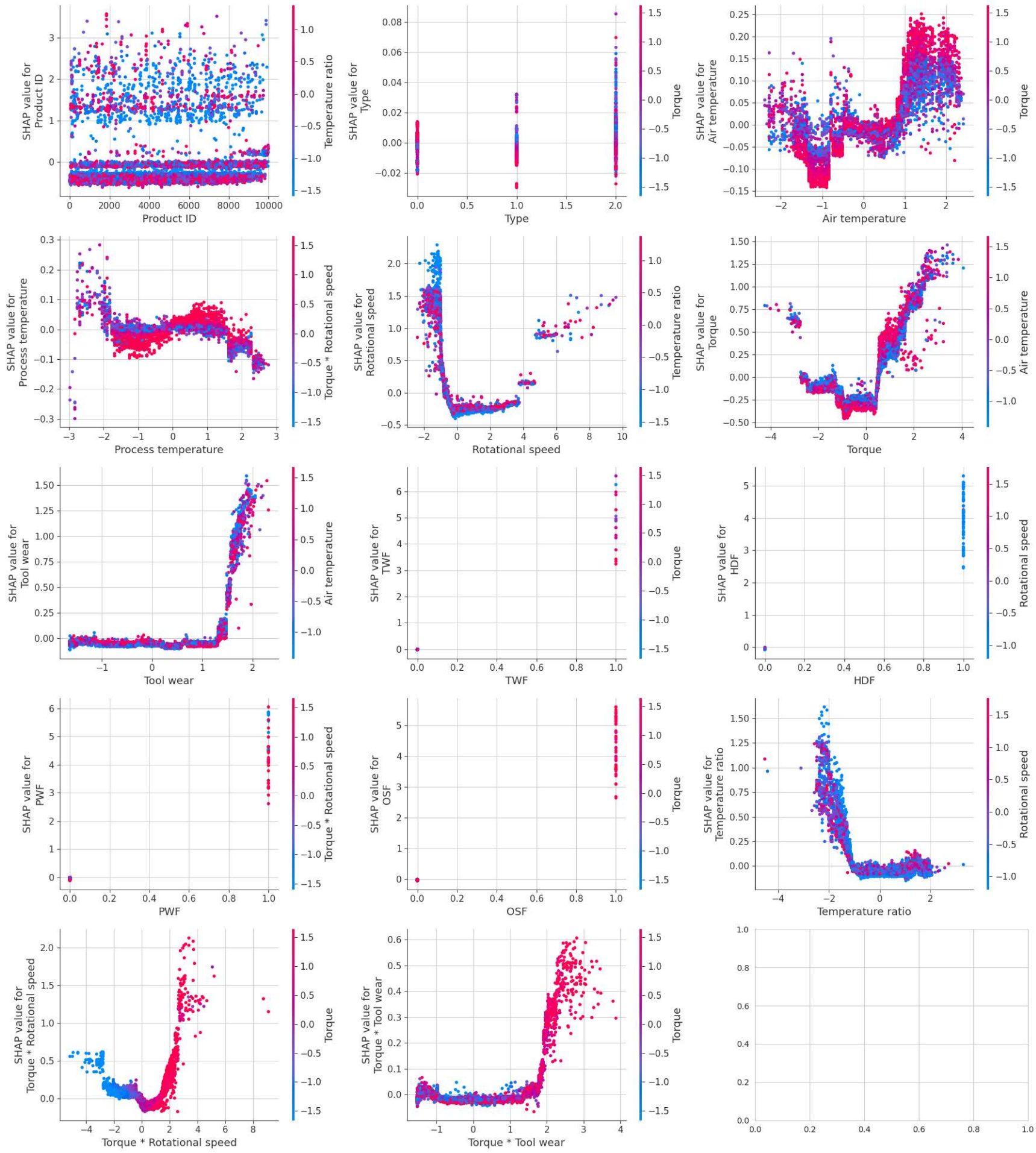
In [34]:
```
# Bar plot
plt.figure(figsize=(20, 14))
shap.summary_plot(shap_values, X_val, plot_type="bar", show=False)
plt.title("Feature Importance - Bar", fontsize=16)

# Dot plot
plt.figure(figsize=(20, 14))
shap.summary_plot(shap_values, X_val, plot_type="dot", show=False)
plt.title("Feature Importance - Dot", fontsize=16)

# Adjust layout and display the plots side by side
plt.tight_layout()
plt.show()
```





In [35]:
```
num_cols = min(3, len(X_val.columns))
num_rows = (len(X_val.columns) + num_cols - 1) // num_cols

fig, axes = plt.subplots(nrows=num_rows, ncols=num_cols, figsize=(6*num_cols, 4*num_rows))

for i, ind in enumerate(X_val.columns):
```

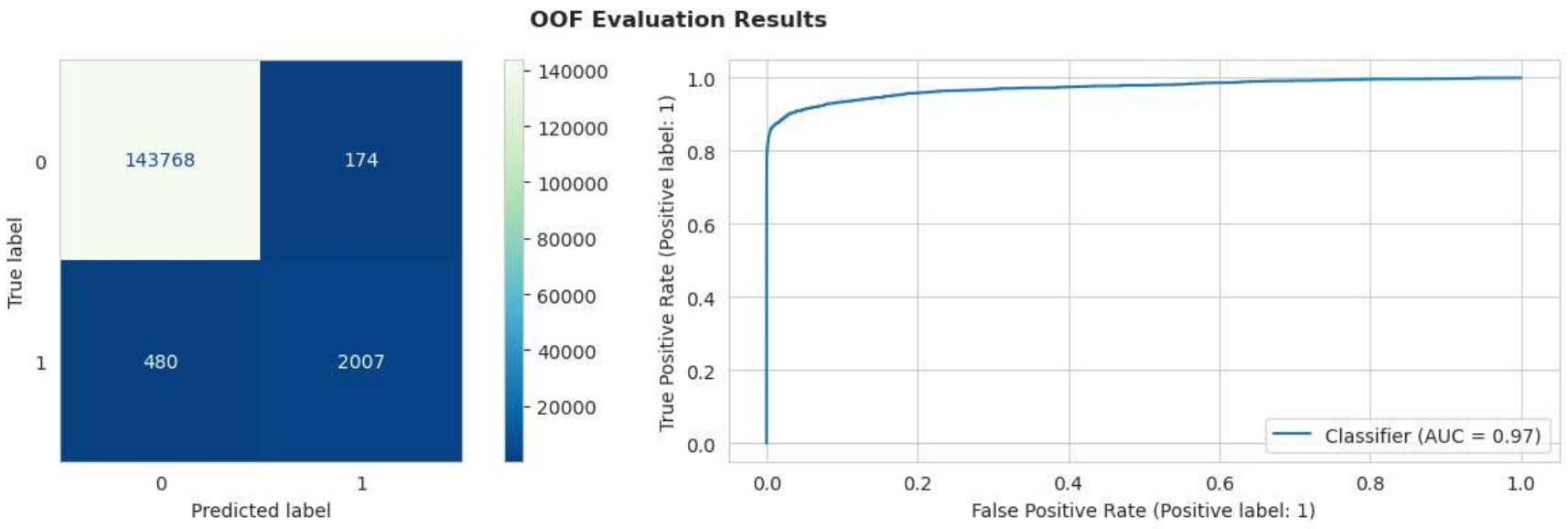
```
    row = i // num_cols
    col = i % num_cols

    shap.dependence_plot(ind=ind, shap_values=shap_values, features=X_val, feature_names=X_val.columns, ax=axes[row, co

plt.tight_layout()
plt.show()
```



Make Submission

In [36]:

```
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

def show_confusion_roc(oof, title='Model Evaluation Results'):
    f, ax = plt.subplots(1, 2, figsize=(13.3, 4))
    df = pd.DataFrame(np.stack([oof[0], oof[1]]), index=['preds', 'target']).T
    cm = confusion_matrix(df.target, df.preds.ge(0.5).astype(int))
    cm_display = ConfusionMatrixDisplay(cm).plot(cmap='GnBu_r', ax=ax[0])
    ax[0].grid(False)
    RocCurveDisplay.from_predictions(df.target, df.preds, ax=ax[1])
    ax[1].grid(True)
    plt.suptitle(f'{title}', fontsize=12, fontweight='bold')
    plt.tight_layout()
    #plt.grid()

show_confusion_roc(oof=[np.mean(oof_predss, axis=1), y_train], title=f'OOF Evaluation Results')
```



In [37]:
```python
def make_submission(test_predss, prefix=''):
    sub = pd.read_csv(os.path.join(filepath, 'sample_submission.csv'))
    sub[f'{target_col}'] = test_predss
    sub.to_csv(f'{prefix}submission.csv', index=False)
    return  sub

sub = make_submission(test_predss, prefix='')
sub
```

Out[37]:

	id	Machine failure
0	136429	0.005779
1	136430	0.012966
2	136431	0.004959
3	136432	0.007903
4	136433	0.006920
...	...	...
90949	227378	0.006422
90950	227379	0.015726
90951	227380	0.005047
90952	227381	0.025733
90953	227382	0.005450

90954 rows × 2 columns

In [38]:
```python
# df_train = pd.read_csv(os.path.join(filepath, 'train.csv'), index_col=[0])
# df_test = pd.read_csv(os.path.join(filepath, 'test.csv'), index_col=[0])
# original = pd.read_csv('/kaggle/input/machine-failure-predictions/machine failure.csv', index_col=[0])

plt.figure(figsize=(16, 6))
sns.set_theme(style="whitegrid")

sns.kdeplot(data=sub, x=target_col, fill=True, alpha=0.5, common_norm=False, label="Predict")
# sns.kdeplot(data=df_train, x=target_col, fill=True, alpha=0.5, common_norm=False, label="Data")
# sns.kdeplot(data=original, x=target_col, fill=True, alpha=0.5, common_norm=False, label="Original")

plt.title('Predictive vs Training Distribution')
plt.legend()
plt.subplots_adjust(top=0.9)
plt.show()
```