DATA SCIENCE

PROJECT

‘Machine Predictive Maintenance’

Submitted By:

ZAUPASH TARIQ (2020-BSE-068)

Submitted To:

Engr. Dr. AAMIR ARSALAN

DEPARTMENT OF SOFTWARE ENGINEERING

FATIMA JINNAH WOMEN UNIVERSITY

Introduction

The focus of this work lies in the field of Data Science, specifically in the application to predictive maintenance. The ability to predict machine failures and identify the nature of these failures is crucial for Industry 4.0. Installing sensors to monitor machine conditions and collecting relevant information can lead to significant cost savings for industries.

The AI4I Predictive Maintenance Dataset from the UCI Repository is utilized for analysis in response to the aforementioned needs. The workflow includes dataset exploration, preprocessing, and the application of machine learning algorithms for two main tasks: predicting machine failures and determining the nature of the failure. The results are compared, considering both performance metrics and interpretability.

1. Tasks and Data Description

Due to the inherent challenges associated with acquiring and sharing authentic predictive maintenance datasets, the UCI repository offers a synthetic dataset crafted to replicate genuine predictive maintenance scenarios encountered in industry. This dataset encompasses 10,000 data points organized in rows, with 14 features structured in columns:

- UID: A unique identifier ranging from 1 to 10000.
- Product ID: Comprising the letters L, M, or H denoting low (60% of all products), medium (30%), and high (10%) product quality variants, along with a variant-specific serial number.
- Air temperature [K]: Generated through a random walk process and subsequently normalized to a standard deviation of 2 K around 300 K.
- Process temperature [K]: Generated using a random walk process normalized to a standard deviation of 1 K, added to the air temperature plus 10 K.
- Rotational speed [rpm]: Calculated from a power of 2860 W, overlaid with normally distributed noise.
- Torque [Nm]: Torque values follow a normal distribution around 40 Nm, with a standard deviation of 10 Nm and no negative values.
- Tool wear [min]: Quality variants H/M/L contribute 5/3/2 minutes of tool wear to the used tool in the process.
- Machine failure: A label indicating whether the machine has failed in a specific data point due to any of the following five independent failure modes:

1. Tool wear failure (TWF): The tool fails or is replaced at a randomly selected tool wear time between 200 - 240 mins.
2. Heat dissipation failure (HDF): Process failure occurs if the difference between air and process temperature is below 8.6 K, and the tool's rotational speed is below 1380 rpm.
3. Power failure (PWF): The product of torque and rotational speed (in rad/s) equals the power required for the process. Process failure happens if this power is below 3500 W or above 9000 W.

4. Overstrain failure (OSF): If the product of tool wear and torque exceeds 11,000 minNm for the L product variant (12,000 M, 13,000 H), the process fails due to overstrain.

5. Random failures (RNF): Each process has a 0.1% chance of failure, irrespective of its process parameters. If at least one of the above failure modes is true, the process fails, and the 'machine failure' label is set to 1. The specific failure mode causing the process failure is not transparent to the machine learning method.

2. Exploratory Analysis

Our initial exploration involves confirming the uniqueness of each entry and the absence of duplicates. This is achieved by verifying that the count of unique ProductID values aligns with the total number of observations. Subsequently, we generate a comprehensive report to identify any missing values and examine the data type of each column.

```
[ ]  import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns

     # Import data
     data = pd.read_csv('predictive_maintenance.csv')
     n = data.shape[0]
     # First checks
     print('Features non-null values and data type:')
     data.info()
     print('Check for duplicate values:',
           data['Product ID'].unique().shape[0]!=n)
```

In summary:

- No data is missing.
- No duplicate values are present.
- Six columns consist of numerical features, including UDI.
- Three columns are categorized as features, including ProductID.

To enhance clarity, we have converted the numeric columns to float type.

```
[ ]  # Set numeric columns dtype to float
     data['Tool wear [min]'] = data['Tool wear [min]'].astype('float64')
     data['Rotational speed [rpm]'] = data['Rotational speed [rpm]'].astype('float64')
     # Rename features
     data.rename(mapper={'Air temperature [K]': 'Air temperature',
                         'Process temperature [K]': 'Process temperature',
                         'Rotational speed [rpm]': 'Rotational speed',
                         'Torque [Nm]': 'Torque',
                         'Tool wear [min]': 'Tool wear'}, axis=1, inplace=True)
```

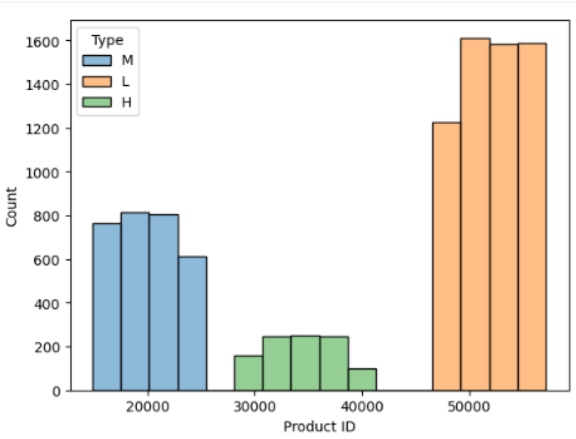
Here are some possible questions:

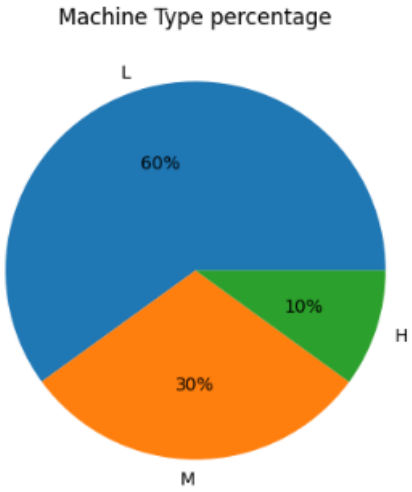
- What is the distribution of machine types (low, medium, high) in the dataset?
- How is the distribution of the target variable (Machine Failure) across different failure modes?

- Can you identify any patterns or trends in the air and process temperature data?
- What is the relationship between rotational speed and torque, and how does it vary across machine types?
- How does tool wear change with the quality variants (low, medium, high)?
- How did the removal of ambiguous and random failure observations impact the dataset?
- What insights can be gained from the boxplots and histograms of numeric features?
- What is the impact of SMOTE resampling on the distribution of failure causes in the dataset?
- How does the distribution of features change after oversampling?
- What is the performance comparison of logistic regression, k-nearest neighbors, support vector machine, and random forest for binary classification?
- Which features contribute the most to the logistic regression model's predictions?
- How does the choice of training, validation, and test datasets affect model performance?
- What is the performance of the logistic regression model for multi-class classification (predicting the nature of failure)?
- How do feature distributions differ when considering different failure modes?
- Are there specific features that are more indicative of certain failure types?
- Can you interpret the decision paths of the models to understand how they MAKE predictions?
- What features are deemed important by decision trees in the random forest model?
- Based on the analysis, what are the key factors influencing machine failure?
- How effective is the chosen resampling technique in addressing class imbalance?

2.1 ID Columns

Before delving into more technical aspects, attention is directed toward the two ID columns, as the model in use might encounter confusion due to them. It is deemed impractical to assume that the failure of a machine is contingent on its identifier. However, it's noteworthy that while the UID aligns with the dataframe index, the Product ID column comprises an initial letter followed by five numbers. There exists a slight possibility that a concealed pattern underlies this structure. Yet, upon closer examination, it is evident that the initial letter corresponds to the machine type, and the numerical sequences delineate three intervals based on the same feature. This confirmation establishes that the Product ID column doesn't provide any additional information beyond the machine type feature, justifying its removal. The ensuing histogram illustrates the number sequences:





Target Anomalies

In this section, we examine the distribution of the target variable to identify any imbalances and address them before proceeding with dataset division. An initial anomaly observed in the dataset pertains to random failures (RNF), where the Machine Failure feature is not set to 1, contrary to the dataset's description.

```
[ ]  # Create lists of features and target names
     features = [col for col in df.columns
                 if df[col].dtype=='float64' or col =='Type']
     target = ['Target','Failure Type']
     # Portion of data where RNF=1
     idx_RNF = df.loc[df['Failure Type']=='Random Failures'].index
     df.loc[idx_RNF,target]
```

Fortunately, occurrences of machine failure due to RNF are limited, totaling only 18 observations. Given the random nature of RNF, making predictions is not feasible. Consequently, we opt to eliminate these rows from the dataset.

```
[ ]  first_drop = df.loc[idx_RNF,target].shape[0]
     print('Number of observations where RNF=1 but Machine failure=0:',first_drop)
     # Drop corresponding observations and RNF column
     df.drop(index=idx_RNF, inplace=True)
```

Subsequently, it is discovered that in 9 observations, Machine Failure is set to 1 when all types of failures are marked as 0. Determining the presence or absence of an actual failure in these instances is challenging. Consequently, we decide to remove these observations as well.

```
[ ]  # Portion of data where Machine failure=1 but no failure cause is specified
     idx_ambiguous = df.loc[(df['Target']==1) &
                            (df['Failure Type']=='No Failure')].index
     second_drop = df.loc[idx_ambiguous].shape[0]
     print('Number of ambiguous observations:', second_drop)
     display(df.loc[idx_ambiguous,target])
     df.drop(index=idx_ambiguous, inplace=True)
```

```
[ ]  # Global percentage of removed observations
     print('Global percentage of removed observations:',
          (100*(first_drop+second_drop)/n))
     df.reset_index(drop=True, inplace=True)   # Reset index
     n = df.shape[0]
```

It is important to note that these actions have not significantly altered the original dataset.

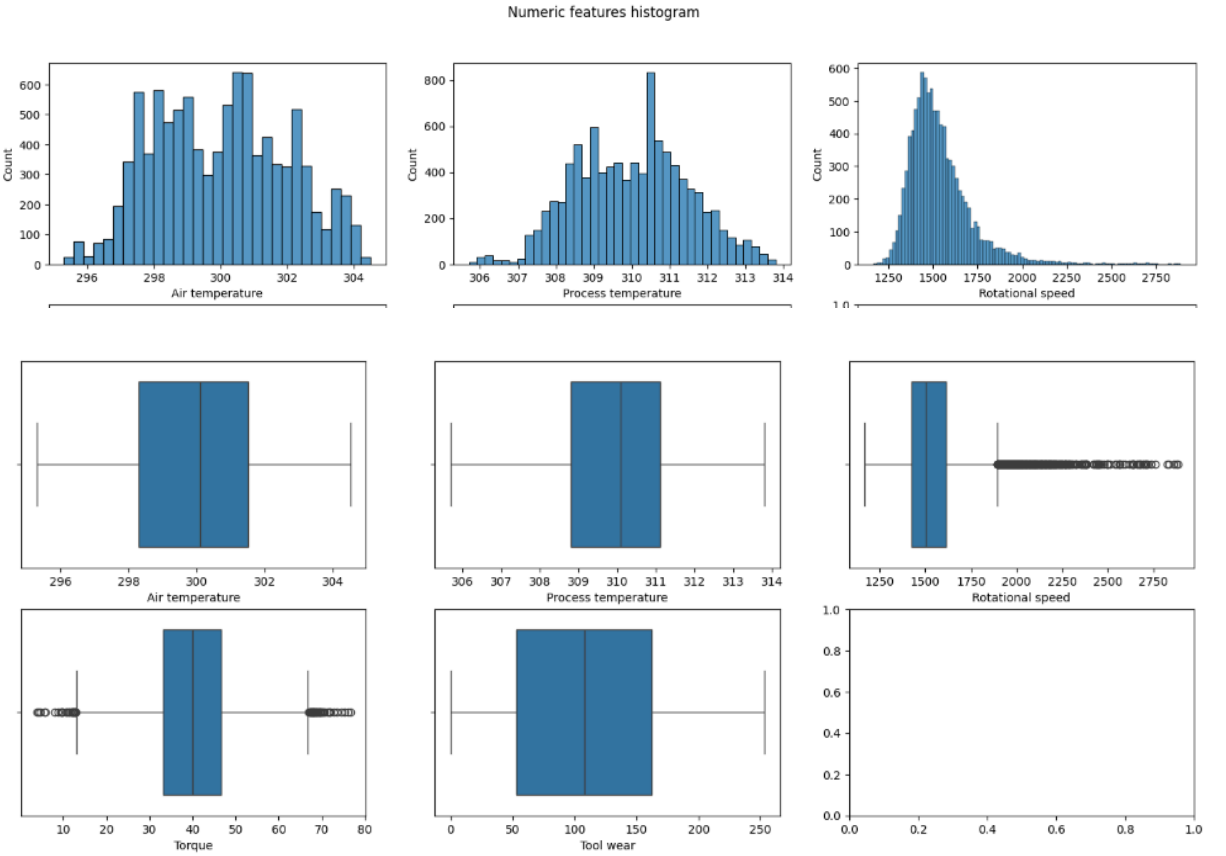
2.3 Outliers Inspection

The objective of this section is to assess the presence of outliers in the dataset, which can often mislead machine learning algorithms. We initiate this examination by delving into a statistical overview of the numerical features.

```
[ ]  df.describe()
```

Upon inspecting the data, indications suggest the potential existence of outliers in Rotational Speed and Torque, as evidenced by the notable difference between the maximum values and the third quartile. To substantiate this observation, we delve deeper into the situation by employing boxplots, coupled with histograms to gain insights into the feature distributions.

```
[ ]  num_features = [feature for feature in features if df[feature].dtype=='float64']
     # Histograms of numeric features
     fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(18,7))
     fig.suptitle('Numeric features histogram')
     for j, feature in enumerate(num_features):
         sns.histplot(ax=axs[j//3, j-3*(j//3)], data=df, x=feature)
     plt.show()

     # boxplot of numeric features
     fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(18,7))
     fig.suptitle('Numeric features boxplot')
     for j, feature in enumerate(num_features):
         sns.boxplot(ax=axs[j//3, j-3*(j//3)], data=df, x=feature)
     plt.show()
```

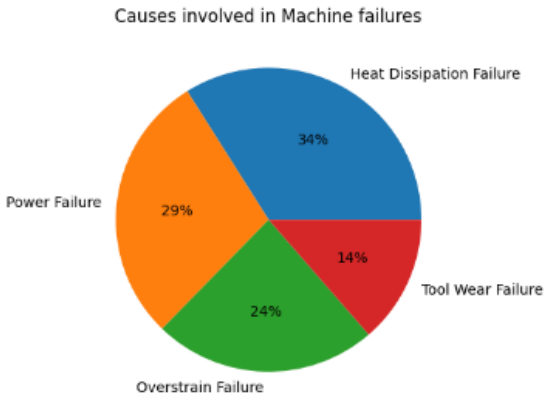


The boxplots indeed bring to light potential outliers in the aforementioned features. However, in the case of Torque, these outliers may be attributable to the method of outlier detection using boxplots. Given the Gaussian distribution, opting for the 3σ rule instead of the Interquartile Range (IQR) might be more appropriate. Conversely, for Rotational Speed, the skewed Gaussian distribution suggests that the few observations with high Rotational Speed may indeed lead to failures. Consequently, we opt to retain the outliers for the time being, reserving the right to decide on further actions after considering additional aspects.

2.4 Resampling with SMOTE

Another crucial consideration pertains to the notably low occurrence of machine failures within the overall dataset, constituting only 3.31%. Furthermore, a pie chart illustrating the distribution of causes for each failure exposes an additional level of imbalance.

```
[ ]  # Portion of df where there is a failure and causes percentage
     idx_fail = df.loc[df['Failure Type'] != 'No Failure'].index
     df_fail = df.loc[idx_fail]
     df_fail_percentage = 100*df_fail['Failure Type'].value_counts()/df_fail['Failure Type'].shape[0]
     print('Failures percentage in data:',
           round(100*df['Target'].sum()/n,2))
     # Pie plot
     plt.title('Causes involved in Machine failures')
     plt.pie(x=df_fail_percentage.array, labels=df_fail_percentage.index.array,
             colors=sns.color_palette('tab10')[0:4], autopct='%.0f%%')
     plt.show()
```



Addressing class imbalance is imperative in machine learning problems, as it can distort both the model training process and our capacity to interpret results accurately. For instance, a model constructed on this dataset predicting that machines never fail could achieve 97% accuracy. To mitigate such effects and curb preferential behavior towards specific classes, we implement data augmentation. The goal is to achieve an 80 to 20 ratio between functioning and faulty observations, maintaining the same percentage distribution among the causes of failures.

Common data augmentation techniques include:

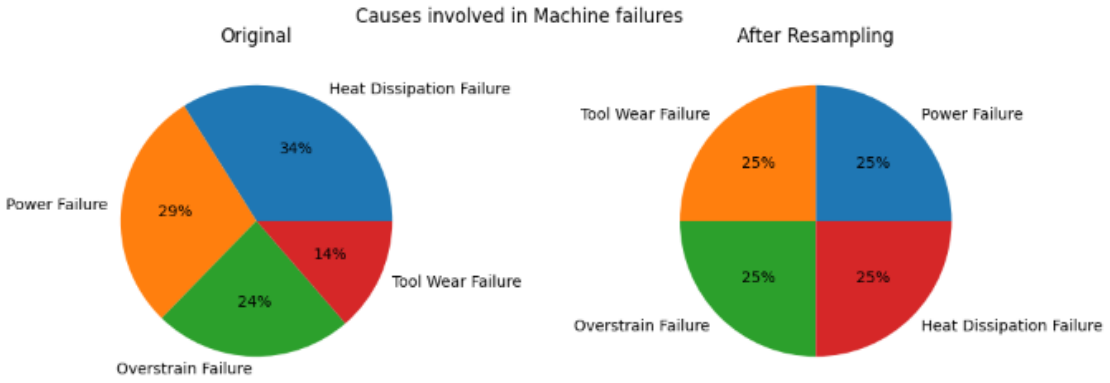
1. Under-sampling by removing some data points from the majority class.
2. Over-sampling by duplicating rows of data in the minority class.
3. Over-sampling with SMOTE (Synthetic Minority Oversampling Technique).

The first two options, however, present overly simplistic approaches. In particular, the first approach diminishes the dataset's length in a context where available data are already limited. Consequently, we opt for the SMOTE procedure to generate new samples, akin to subtly shifting data points in the direction of their neighbors. This ensures that synthetic data points are not exact copies but also not drastically different from known observations in the minority class. To elaborate, the SMOTE procedure involves randomly selecting a sample from the minority class, identifying k nearest neighbors for the observations in this sample, selecting one of those neighbors, determining the vector between the current data point and the chosen neighbor, multiplying the vector by a random number between 0 and 1, and obtaining the synthetic data point by adding this scaled vector to the current data point.

2.5 Comparison after resampling

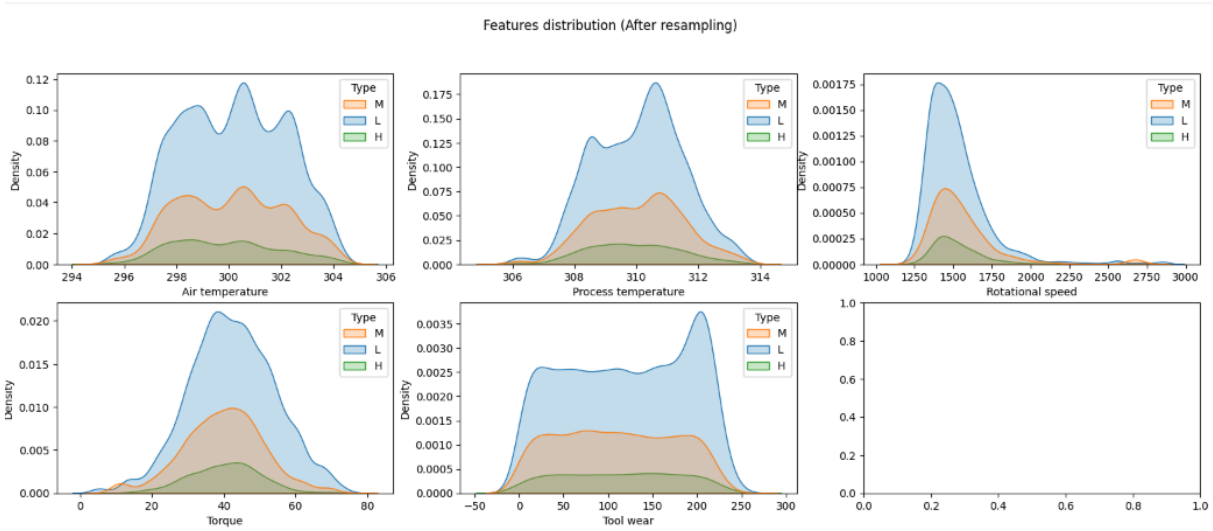
Results are described in following Pie Charts:

```
[ ] from sklearn.model_selection import train_test_split
    from imblearn.over_sampling import SMOTENC
    # n_working must represent 80% of the desired length of resampled dataframe
    n_working = df['Failure Type'].value_counts()['No Failure']
    desired_length = round(n_working/0.8)
    spc = round((desired_length-n_working)/4)  #samples per class
    # Resampling
    balance_cause = {'No Failure':n_working,
                     'Overstrain Failure':spc,
                     'Heat Dissipation Failure':spc,
                     'Power Failure':spc,
                     'Tool Wear Failure':spc}
    sm = SMOTENC(categorical_features=[0,7], sampling_strategy=balance_cause, random_state=0)
    df_res, y_res = sm.fit_resample(df, df['Failure Type'])
```



As anticipated, instances of machine failure are predominantly associated with low-quality machines, followed by those of medium quality, and relatively few cases involve high-quality machines. This distinction becomes more pronounced when the number of non-functioning machine observations is artificially increased. However, the kernel density estimate (kdeplots) below reveals that this correlation is not significantly linked to specific features. When differentiating based on machine quality, the distribution of features shows minimal variations, except for the two side peaks in Tool Wear, aligning with the data description. This suggests that the predominance of failures in type L machines is likely attributable to the higher representation of this type in the dataset. Thus, the correlation with machine failure appears to be driven by statistical factors.

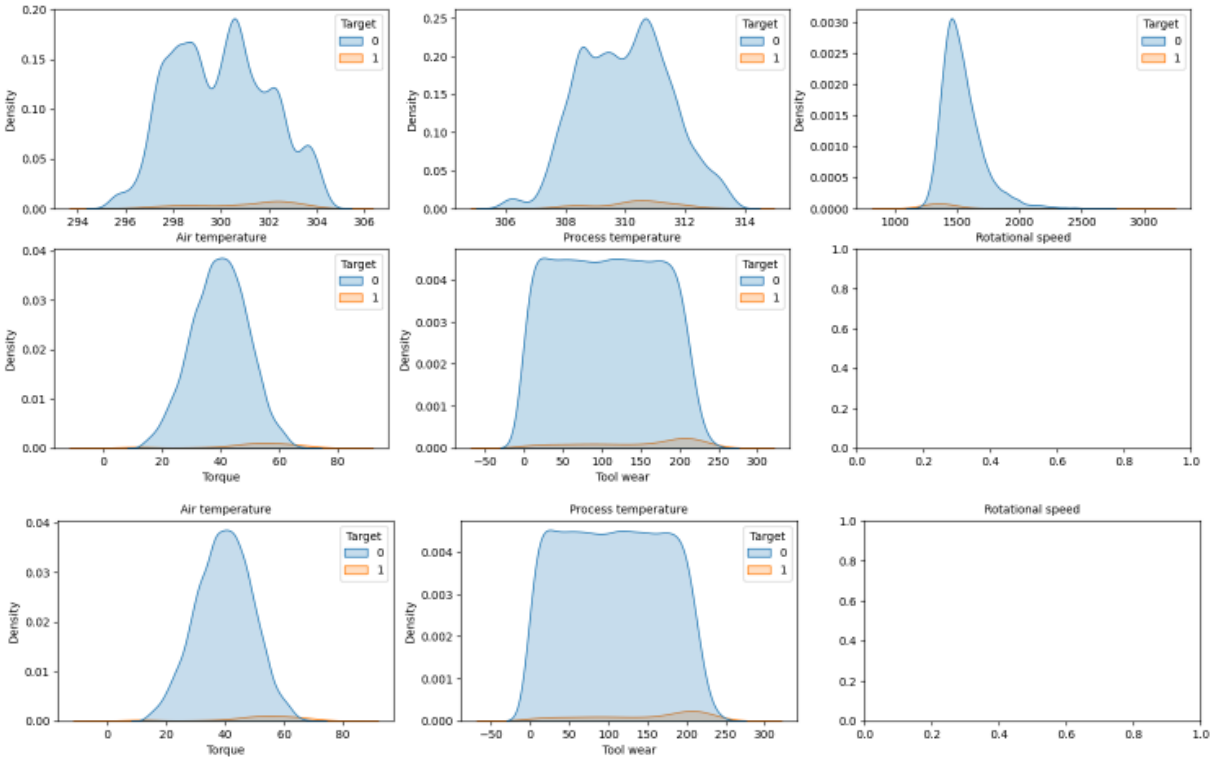
```
# Kdeplot of numeric features (After resampling) - hue=Type
fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(19,7))
fig.suptitle('Features distribution (After resampling)')
custom_palette = {'L':'tab:blue', 'M':'tab:orange', 'H':'tab:green'}
for j, feature in enumerate(num_features):
    sns.kdeplot(ax=axs[j//3, j-3*(j//3)], data=df_res, x=feature,
                hue='Type', fill=True, palette=custom_palette)
plt.show()
```



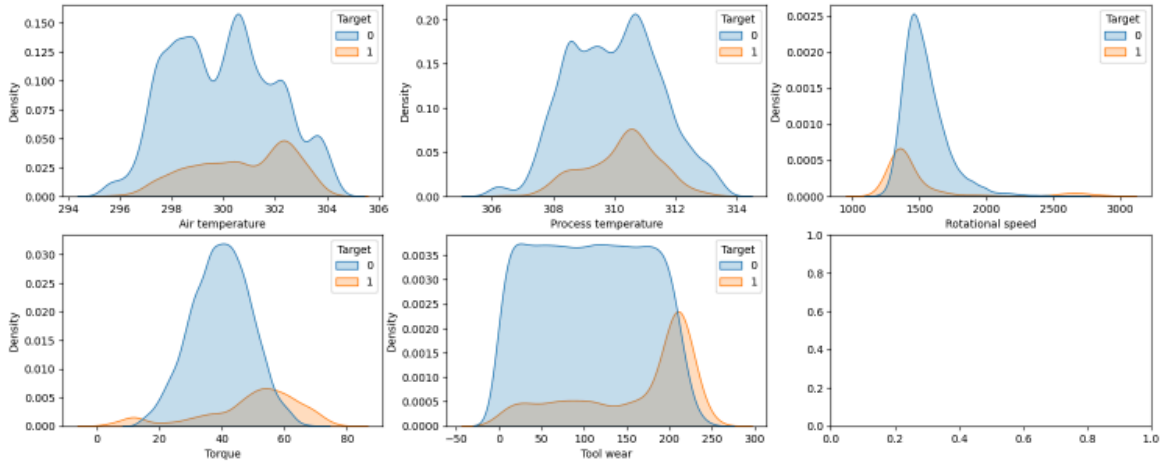
Finally, let's look at how the distribution of features has changed.

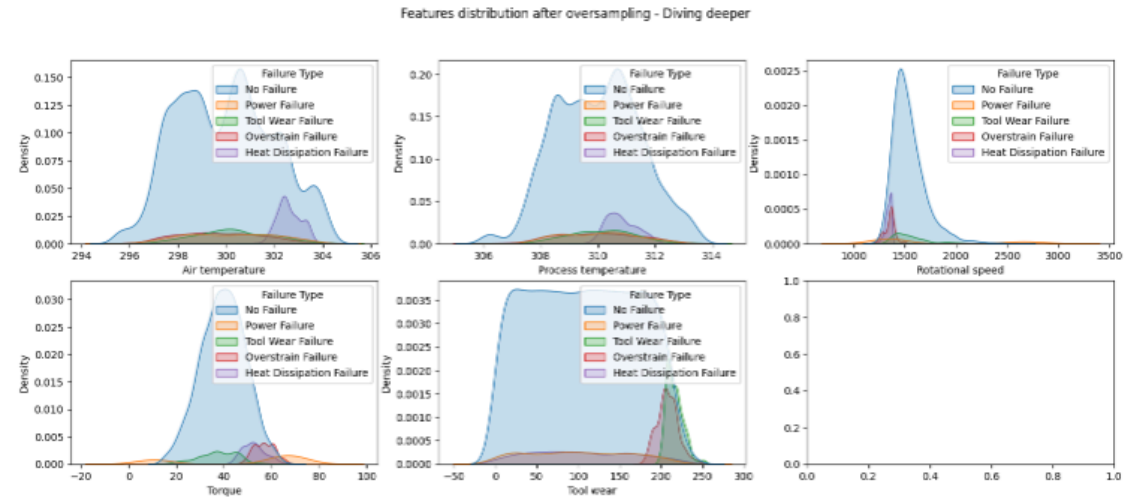
```
# Kdeplot of numeric features (Original)
fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(18,7))
fig.suptitle('Original Features distribution')
enumerate_features = enumerate(num_features)
for j, feature in enumerate_features:
    sns.kdeplot(ax=axs[j//3, j-3*(j//3)], data=df, x=feature,
                hue='Target', fill=True, palette='tab10')
plt.show()
# Kdeplot of numeric features (After resampling)
fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(18,7))
fig.suptitle('Features distribution after oversampling')
enumerate_features = enumerate(num_features)
for j, feature in enumerate_features:
    sns.kdeplot(ax=axs[j//3, j-3*(j//3)], data=df_res, x=feature,
                hue=df_res['Target'], fill=True, palette='tab10')
plt.show()
# Kdeplot of numeric features (After resampling) - Diving deeper
fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(18,7))
fig.suptitle('Features distribution after oversampling - Diving deeper')
enumerate_features = enumerate(num_features)
for j, feature in enumerate_features:
    sns.kdeplot(ax=axs[j//3, j-3*(j//3)], data=df_res, x=feature,
                hue=df_res['Failure Type'], fill=True, palette='tab10')
plt.show()
```

Original Features distribution



Features distribution after oversampling





An initial observation reveals the successful execution of data augmentation, as the feature distribution for faulty instances has not undergone significant distortion. Notably, in Rotational Speed, Torque, and Tool Wear, observations related to failures exhibit a density peak in extreme zones of the distribution. This suggests that the outliers discussed in Section 2.3 are not attributable to errors in dataset construction but rather reflect the inherent variance within the data. Further clarity emerges when examining the distributions specific to individual causes of failure. Particularly, a nearly symmetrical behavior is evident in Rotational Speed and Torque. In contrast, Tool Wear demonstrates a distinct separation between PWF and HDF failures at lower values, with peaks identified at higher values corresponding to TWF and OSF. This alignment aligns seamlessly with the target descriptions provided in the "Task and dataset description" section.

2.6 Features Scaling and Encoding

To render the data suitable for the forthcoming algorithms, we implement two transformations:

Initially, we employ label encoding on the categorical columns. Since "Type" is an ordinal feature and "Cause" needs to be represented in a single column, we map them as follows:
- Type: {L=0, M=1, H=2}
- Cause: {Working=0, PWF=1, OSF=2, HDF=3, TWF=4}

Subsequently, we execute column scaling using StandardScaler. This step proves particularly beneficial for the optimal functioning of methods reliant on the metric space, such as PCA and KNN. Additionally, it has been confirmed that utilizing StandardScaler yields marginally superior performances compared to MinMaxScaler.

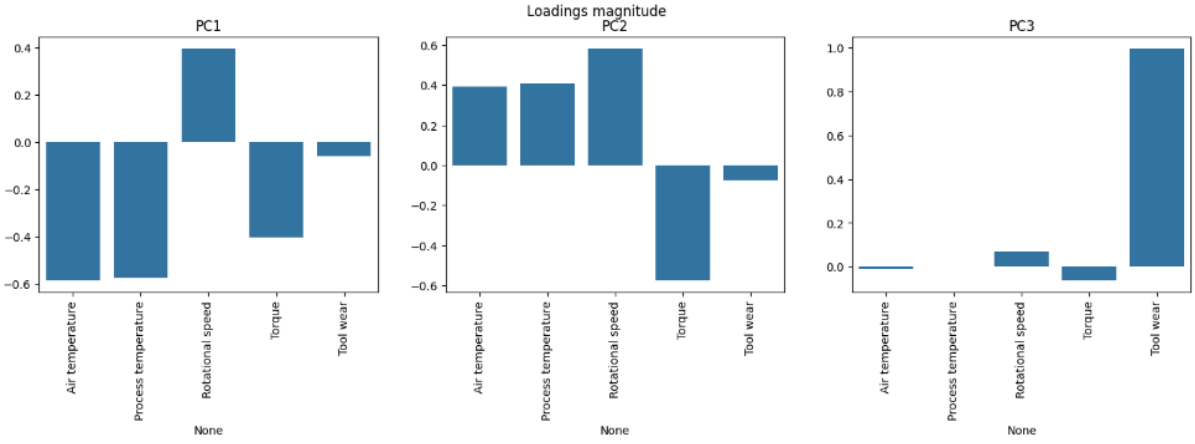
```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

sc = StandardScaler()
type_dict = {'L': 0, 'M': 1, 'H': 2}
cause_dict = {'No Failure': 0,
              'Power Failure': 1,
              'Overstrain Failure': 2,
              'Heat Dissipation Failure': 3,
              'Tool Wear Failure': 4}
df_pre = df_res.copy()
# Encoding
df_pre['Type'].replace(to_replace=type_dict, inplace=True)
df_pre['Failure Type'].replace(to_replace=cause_dict, inplace=True)
# Scaling
df_pre[num_features] = sc.fit_transform(df_pre[num_features])
```

2.7 PCA and Correlation Heatmap

We run PCA to have a further way of displaying the data instead of making feature selection.

```
[ ]  pca = PCA(n_components=len(num_features))
     X_pca = pd.DataFrame(data=pca.fit_transform(df_pre[num_features]), columns=['PC'+str(i+1) for i in range(len(num_features))])
     var_exp = pd.Series(data=100*pca.explained_variance_ratio_, index=['PC'+str(i+1) for i in range(len(num_features))])
     print('Explained variance ratio per component:', round(var_exp,2), sep='\n')
     print('Explained variance ratio with 3 components: '+str(round(var_exp.values[:3].sum(),2)))
```

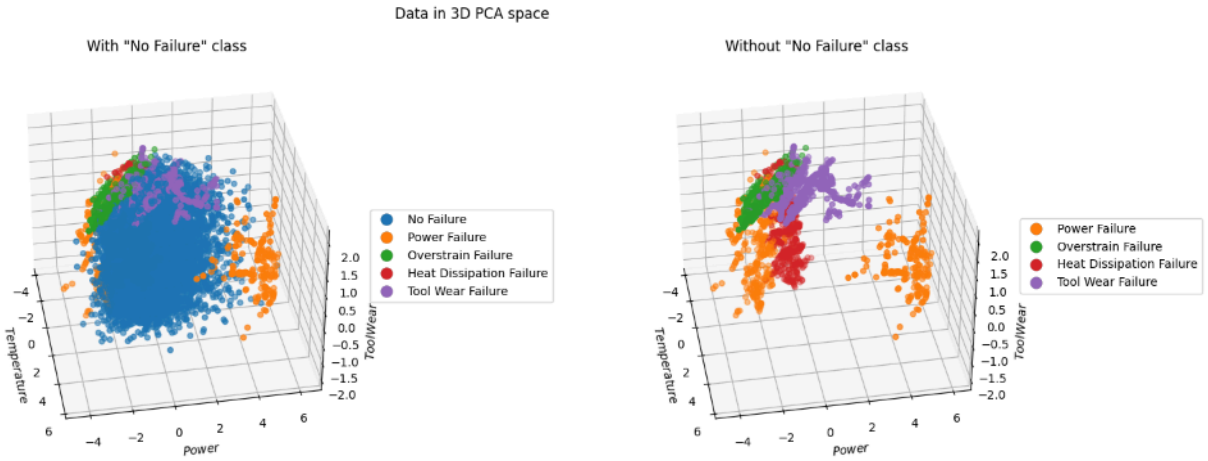
Since the first three components are enough to almost fully represent the variance of the data we will project them in a three dimensional space.



The bar plot illustrating the weights of Principal Components provides a clear insight into their respective representations:

- PC1 is strongly associated with the two temperature variables.

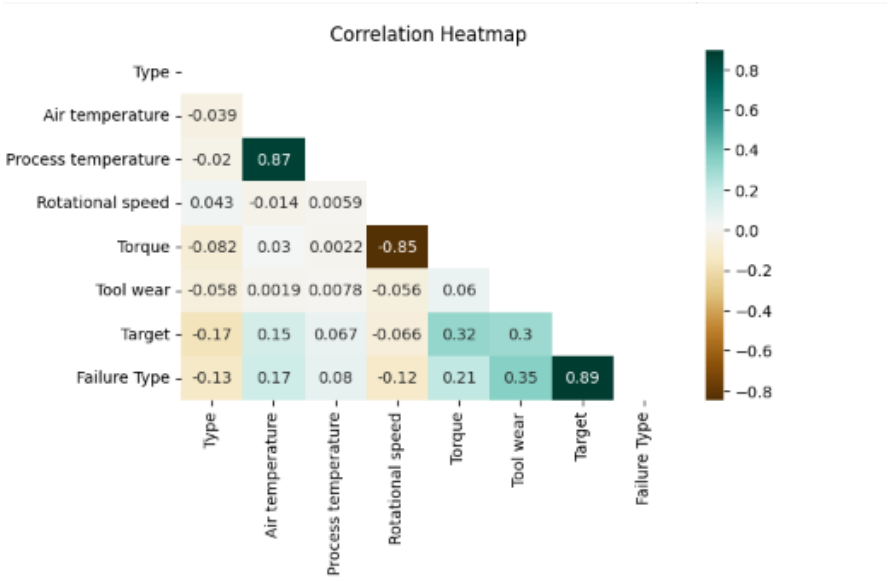
- PC2 can be linked to the machine power, which results from the product of Rotational Speed and Torque.
- PC3 is distinctly related to Tool Wear.



The representation in the space defined by these three axes underscores the following observations:

1. TWF emerges as the failure class most distinctly isolated from the others, primarily reliant on PC3 (Tool Wear).
2. PWF spans two distinct bands along PC2 (Power), demonstrating independence from the other two components.
3. The separation between OSF and HDF classes is less pronounced compared to others, although noticeable distinctions are observed. OSF tends to exhibit high Tool Wear and low power, while HDF is characterized by elevated temperature and low power.

```
# Correlation Heatmap
plt.figure(figsize=(7,4))
sns.heatmap(data=df_pre.corr(), mask=np.triu(df_pre.corr()), annot=True, cmap='BrBG')
plt.title('Correlation Heatmap')
plt.show()
```



As expected, we note a significant correlation among temperature-related features and those associated with power. Additionally, Tool Wear exhibits a robust correlation with both of our target variables, validating our earlier observations from the PCA analysis. Lastly, a somewhat less pronounced correlation is evident between torque and the two target variables.

2.8 Metrics

To quantitatively assess the performance of the models, we rely on several metrics that encapsulate key aspects of the classification results:

1. Accuracy
2. AUC (Area Under the Curve)
3. F1 Score

3. Binary Task

3.1 Preliminaries

The objective of this section is to identify the optimal model for binary classification in predicting the occurrence of Machine Failure in the dataset. Classification algorithms, integral to data mining, leverage supervised machine learning techniques to make predictions about data. Specifically, these algorithms utilize labeled datasets, divided into two or more classes, to create a classification model. This model is then applied to new, unlabeled data to assign them to the appropriate class. The initial dataset is typically segmented into three groups: the training dataset (used to fit the model), the validation dataset (employed to evaluate model fit while tuning hyperparameters), and the test dataset (utilized for model testing).

In this project, we adopt a split ratio of 80/10/10 for the dataset division. This ratio is chosen after testing various strategies, demonstrating its efficacy. The classification techniques implemented are as follows:

1.**Logistic Regression:** This method estimates the probability of a dependent variable as a function of independent variables. It serves as a benchmark model due to its simplicity and interpretability, providing a foundational comparison point for evaluating other models' results.

2. **K-nearest neighbors (K-NN)**: This algorithm calculates the distance between dataset elements and assigns data to a specific class if it is close enough to other data of the same class. The parameter K represents the number of neighboring data considered when assigning classes.

3**. Support Vector Machine (SVM):** SVM aims to find a hyperplane in an N-dimensional space (N—number of features) that distinctly classifies data points while maximizing the margin distance, i.e., the distance between data points of both classes.

4. **Random Forest:** This ensemble learning technique combines multiple classifiers to address complex problems. Random Forest, utilizing the bagging technique, constructs numerous decision trees in parallel, all with equal importance. The final output is determined by the class selected by the majority of trees.

```
from sklearn.metrics import accuracy_score, roc_auc_score, f1_score, fbeta_score
from sklearn.metrics import confusion_matrix, make_scorer
from sklearn.inspection import permutation_importance
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.svm import SVC
import time

# train-validation-test split
X, y = df_pre[features], df_pre[['Target','Failure Type']]
X_trainval, X_test, y_trainval, y_test = train_test_split(X, y, test_size=0.1, stratify=df_pre['Failure Type'], random_state=0)
X_train, X_val, y_train, y_val = train_test_split(X_trainval, y_trainval, test_size=0.11, stratify=y_trainval['Failure Type'], random_state=0)
```

We define some functions to make the following subsections easier to read. If not interested in the details we suggest to skip to Section 3.1

```
"""User-defined function: Evaluate cm, accurcay, AUC, F1 for a given classifier
- model, fitted estimator.
- X, data used to estimate class probabilities (paired with y_true)
- y_true, ground truth with two columns
- y_pred, predictions
- task = 'binary','multi_class'
"""
def eval_preds(model,X,y_true,y_pred,task):
    if task == 'binary':
        # Extract task target
        y_true = y_true['Target']
        cm = confusion_matrix(y_true, y_pred)
        # Probability of the minority class
        proba = model.predict_proba(X)[:,1]
        # Metrics
        acc = accuracy_score(y_true, y_pred)
        auc = roc_auc_score(y_true, proba)
        f1 = f1_score(y_true, y_pred, pos_label=1)
        f2 = fbeta_score(y_true, y_pred, pos_label=1, beta=2)
    elif task == 'multi_class':
        y_true = y_true['Failure Type']
```

```
        cm = confusion_matrix(y_true, y_pred)
        proba = model.predict_proba(X)
        # Metrics
        acc = accuracy_score(y_true, y_pred)
        auc = roc_auc_score(y_true, proba, multi_class='ovr', average='weighted')
        f1 = f1_score(y_true, y_pred, average='weighted')
        f2 = fbeta_score(y_true, y_pred, beta=2, average='weighted')
    metrics = pd.Series(data={'ACC':acc, 'AUC':auc, 'F1':f1, 'F2':f2})
    metrics = round(metrics,3)
    return cm, metrics



"""User-defined function: Fits one estimator using GridSearch to search for the best parameters
- clf, estimator
- X, y = X_train, y_train
- params, parameters grid for GridSearch
- task = 'binary','multi_class'
"""
def tune_and_fit(clf,X,y,params,task):
    if task=='binary':
        f2_scorer = make_scorer(fbeta_score, pos_label=1, beta=2)
        start_time = time.time()
        grid_model = GridSearchCV(clf, param_grid=params,
                                  cv=5, scoring=f2_scorer)
        grid_model.fit(X, y['Target'])
    elif task=='multi_class':
        f2_scorer = make_scorer(fbeta_score, beta=2, average='weighted')
        start_time = time.time()
```

```
        f2_scorer = make_scorer(fbeta_score, beta=2, average='weighted')
        start_time = time.time()
        grid_model = GridSearchCV(clf, param_grid=params,
                                  cv=5, scoring=f2_scorer)
        grid_model.fit(X, y['Failure Type'])

    print('Best params:', grid_model.best_params_)
    # Print training times
    train_time = time.time()-start_time
    mins = int(train_time//60)
    print('Training time: '+str(mins)+'m '+str(round(train_time-mins*60))+'s')
    return grid_model



"""User-defined function: Makes predictions using the tuned classifiers.
Then uses eval_preds to compute the relative metrics. Returns:
- y_pred, DataFrame containing the predictions of each model
- cm_list, confusion matrix list
- metrics, DataFrame containing the metrics
Input:
- fitted_models, fitted estimators
- X, data used to make predictions
- y_true, true values for target
- clf_str, list containing estimators names
- task = 'binary','multi_class'
"""
def predict_and_evaluate(fitted_models,X,y_true,clf_str,task):
    cm_dict = {key: np.nan for key in clf_str}
    metrics = pd.DataFrame(columns=clf_str)
    y_pred = pd.DataFrame(columns=clf_str)
    for fit_model, model_name in zip(fitted_models,clf_str):
        # Update predictions
```

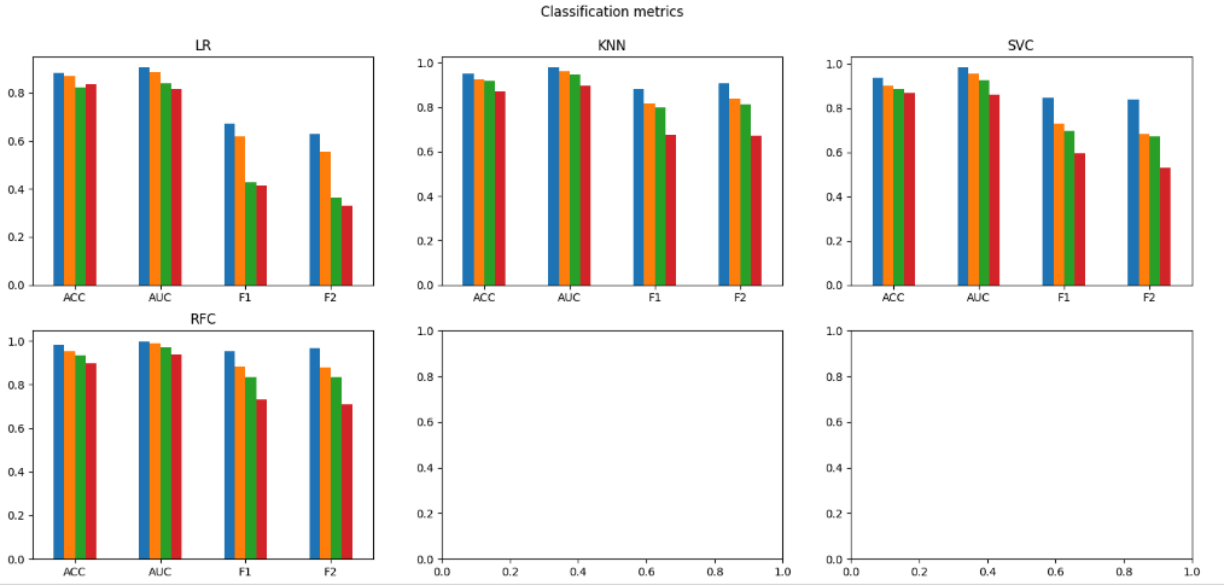
```
        y_pred[model_name] = fit_model.predict(X)
        # Metrics
        if task == 'binary':
            cm, scores = eval_preds(fit_model,X,y_true,
                                    y_pred[model_name],task)
        elif task == 'multi_class':
            cm, scores = eval_preds(fit_model,X,y_true,
                                    y_pred[model_name],task)
        # Update Confusion matrix and metrics
        cm_dict[model_name] = cm
        metrics[model_name] = scores
    return y_pred, cm_dict, metrics



"""User-defined function: Fit the estimators on multiple classifiers
- clf, estimators
- clf_str, list containing estimators names
- X_train,y_train, data used to fit models
- X_val,y_val, data used to validate models
"""

def fit_models(clf,clf_str,X_train,X_val,y_train,y_val):
    metrics = pd.DataFrame(columns=clf_str)
    for model, model_name in zip(clf, clf_str):
        model.fit(X_train,y_train['Target'])
        y_val_pred = model.predict(X_val)
        metrics[model_name] = eval_preds(model,X_val,y_val,y_val_pred,'binary')[1]
    return metrics
```

3.2 Feature selection attempts

Before delving into the model training phase, we explore feature selection, leveraging insights from the correlation heatmap and exploratory data analysis. As a reminder, we observed positive correlation between "Process temperature" and "Air temperature," as well as negative correlation between "Torque" and "Rotational speed." According to the dataset description, the occurrence of Power Failure (PWF) is linked to the product of "Torque" and "Rotational speed," while Heat Dissipation Failure (HDF) is related to the difference between "Air temperature" and "Process temperature."

Considering these correlations, outright removal of these columns might result in significant information loss. Hence, we opt to examine the impact of combining these features, pairwise, to create new features while preserving their physical interpretation. The comparison involves fitting classification models without tuning any parameters on the following datasets:

1. The original dataset.
2. A dataset obtained by removing the "Process temperature" and "Air temperature" columns and replacing them with a column representing their product.
3. A dataset obtained by removing "Torque" and "Rotational speed" and replacing them with a column representing their product.
4. A dataset resulting from a combination of the aforementioned operations.



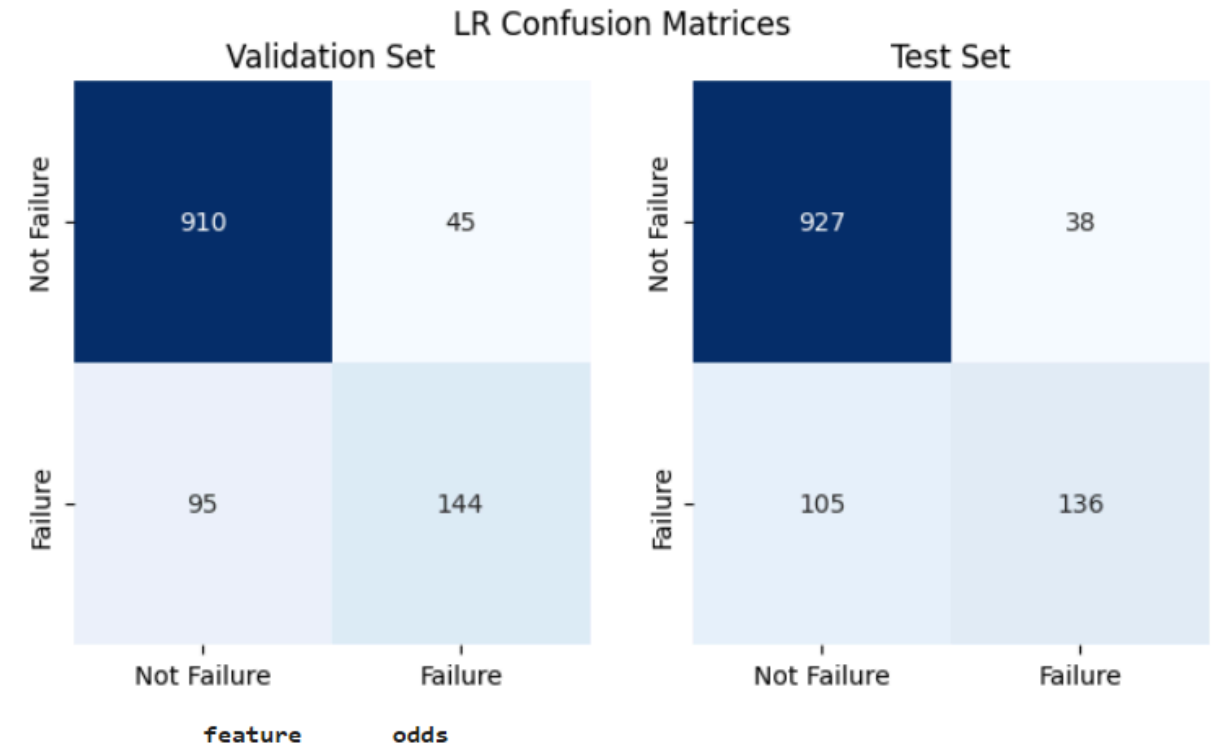
Based on the achieved results, it is evident that the models applied to the complete dataset exhibit superior performance compared to their application on datasets with reduced features. The optimal performance coupled with the relatively concise number of features in our dataset motivates the decision to forego the feature selection step.

3.3. Logistic Regression Benchmark

We decide to use Logistic Regression as a Benchmark for our task. It represents an intermediate step between the basic model referred to in Section 2.4 and the more complex models that we have described and we will explore in depth in the following sections. Now we look at the results obtained and at the interpretability of the model

```
# Make predictions
lr = LogisticRegression(random_state=0)
lr.fit(X_train, y_train['Target'])
y_val_lr = lr.predict(X_val)
y_test_lr = lr.predict(X_test)

# Metrics
cm_val_lr, metrics_val_lr = eval_preds(lr,X_val,y_val,y_val_lr,'binary')
cm_test_lr, metrics_test_lr = eval_preds(lr,X_test,y_test,y_test_lr,'binary')
print('Validation set metrics:',metrics_val_lr, sep='\n')
print('Test set metrics:',metrics_test_lr, sep='\n')

cm_labels = ['Not Failure', 'Failure']
cm_lr = [cm_val_lr, cm_test_lr]
# Show Confusion Matrices
fig, axs = plt.subplots(ncols=2, figsize=(8,4))
fig.suptitle('LR Confusion Matrices')
for j, title in enumerate(['Validation Set', 'Test Set']):
    ax = axs[j]
    sns.heatmap(ax=ax, data=cm_lr[j], annot=True,
                fmt='d', cmap='Blues', cbar=False)
    axs[j].title.set_text(title)
    axs[j].set_xticklabels(cm_labels)
    axs[j].set_yticklabels(cm_labels)
plt.show()

# Odds for interpretation
d = {'feature': X_train.columns, 'odds': np.exp(lr.coef_[0])}
odds_df = pd.DataFrame(data=d).sort_values(by='odds', ascending=False)
odds_df
```

```
Validation set metrics:
ACC    0.883
AUC    0.905
F1     0.673
F2     0.629
dtype: float64
Test set metrics:
ACC    0.881
AUC    0.917
F1     0.655
F2     0.598
dtype: float64
```




```
feature      odds
```

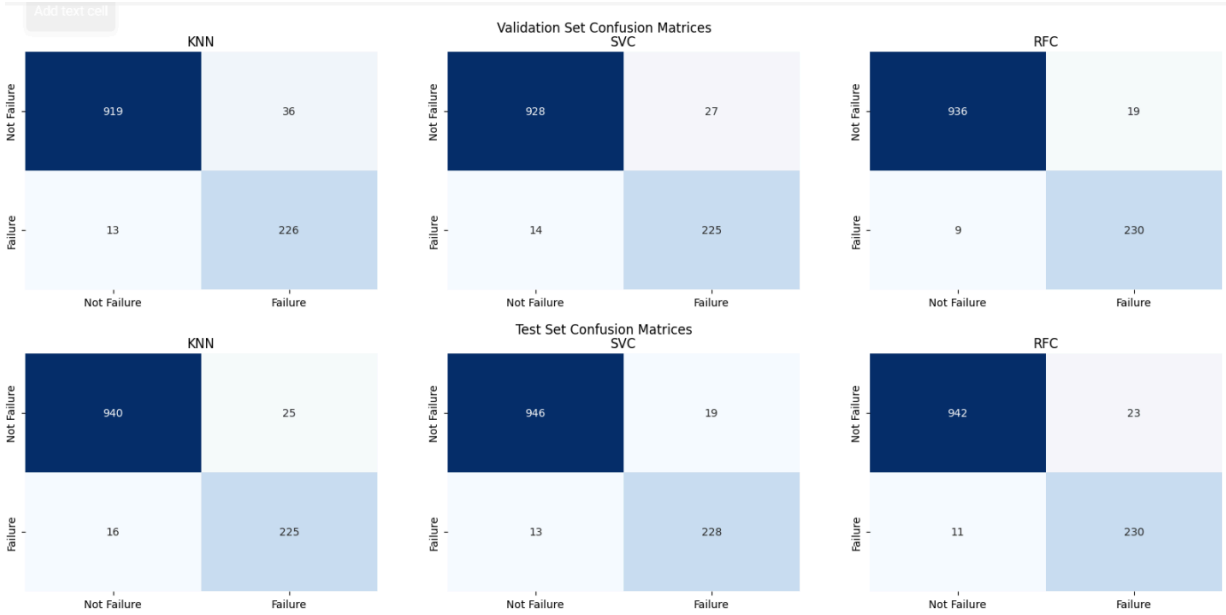
The odds of logistic regression allow us to understand how the model is working. In particular, an unrealistically high importance is given to Torque and Rotational Speed. This is mainly due to the natural variance in these features, which is especially high when looking only at the failure cases and tends to "deviate" the model. However it is reasonable to believe, on the basis of exploratory analysis, that the first four features have a significantly greater relevance than the last two. We also expect greater reliability of the odds values when we apply logistic regression to the multiclass task, since the effects that are spread here appears to be localized around certain types of failures.

3.4.Models

```python
# Models
knn = KNeighborsClassifier()
svc = SVC()
rfc = RandomForestClassifier()
clf = [knn,svc,rfc]
clf_str = ['KNN','SVC','RFC']

# Parameter grids for GridSearch
knn_params = {'n_neighbors':[1,3,5,8,10]}
svc_params = {'C': [1, 10, 100],
              'gamma': [0.1,1],
              'kernel': ['rbf'],
              'probability':[True],
              'random_state':[0]}
rfc_params = {'n_estimators':[100,300,500,700],
              'max_depth':[5,7,10],
              'random_state':[0]}
params = pd.Series(data=[knn_params,svc_params,rfc_params],
                   index=clf)

# Tune hyperparameters with GridSearch (estimated time 8m)
print('GridSearch start')
fitted_models_binary = []
for model, model_name in zip(clf, clf_str):
    print('Training '+str(model_name))
    fit_model = tune_and_fit(model,X_train,y_train,params[model],'binary')
    fitted_models_binary.append(fit_model)
```

```
GridSearch start
Training KNN
Best params: {'n_neighbors': 1}
Training time: 0m 4s
Training SVC
Best params: {'C': 100, 'gamma': 1, 'kernel': 'rbf', 'probability': True, 'random_state': 0}
Training time: 1m 57s
Training RFC
Best params: {'max_depth': 10, 'n_estimators': 500, 'random_state': 0}
Training time: 3m 29s
```

```
# Create evaluation metrics
task = 'binary'
y_pred_val, cm_dict_val, metrics_val = predict_and_evaluate(
    fitted_models_binary,X_val,y_val,clf_str,task)
y_pred_test, cm_dict_test, metrics_test = predict_and_evaluate(
    fitted_models_binary,X_test,y_test,clf_str,task)

# Show Validation Confusion Matrices
fig, axs = plt.subplots(ncols=3, figsize=(20,4))
fig.suptitle('Validation Set Confusion Matrices')
for j, model_name in enumerate(clf_str):
    ax = axs[j]
    sns.heatmap(ax=ax, data=cm_dict_val[model_name], annot=True,
                fmt='d', cmap='Blues', cbar=False)
    ax.title.set_text(model_name)
    ax.set_xticklabels(cm_labels)
    ax.set_yticklabels(cm_labels)
plt.show()

# Show Test Confusion Matrices
fig, axs = plt.subplots(ncols=3, figsize=(20,4))
fig.suptitle('Test Set Confusion Matrices')
for j, model_name in enumerate(clf_str):
    ax = axs[j]
    sns.heatmap(ax=ax, data=cm_dict_test[model_name], annot=True,
                fmt='d', cmap='Blues', cbar=False)
    ax.title.set_text(model_name)
    ax.set_xticklabels(cm_labels)
    ax.set_yticklabels(cm_labels)
plt.show()

# Print scores
print('')
print('Validation scores:', metrics_val, sep='\n')
print('Test scores:', metrics_test, sep='\n')
```



Validation scores:

	KNN	SVC	RFC
ACC	0.959	0.966	0.977
AUC	0.954	0.987	0.997
F1	0.902	0.916	0.943
F2	0.928	0.931	0.954

Test scores:

	KNN	SVC	RFC
ACC	0.966	0.973	0.972
AUC	0.954	0.992	0.997
F1	0.916	0.934	0.931
F2	0.927	0.941	0.945

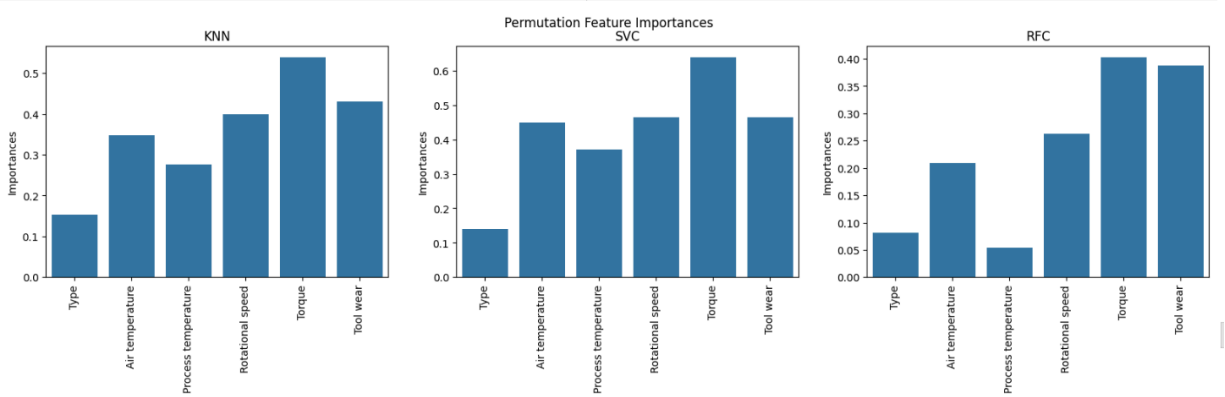
All the selected models obtain similar results on the validation set (except KNN which is a little worse) and it is diffj7icult to determine if one works better than another by looking only at these values. Performance did not significantly drop when passing the test set, showing that overfitting was avoided. We comment on the results of the models by looking at the confusion matrices and the metrics obtained on the test set: in this way the formation of a hierarchy between the models used is slightly clearer, as all the metrics relating to a single model are smaller or larger than to the others and the time needed to search for the parameters is comparable, with the only exception of KNN. In particular KNN obtains the worst performances; in the middle we find SVC and RFC which achieve extremely similar results.

About the parameters:

- A Gridsearch has been started on the parameters which, looking in the literature, appear to be preponderant for each specific model;
- The grid values to search for have been defined on the basis of literature and various tests, trying to keep the computational cost of finding the best values moderate.

It is interesting to observe that the optimal parameters for RFC and XGB are the polar opposite: the former prefers to use a few estimators and go into depth while the latter uses more estimators with fewer splits. Furthermore, it must be taken into account that although XGB is the best classifier from a quantitative point of view, this is not true for what concerns the qualitative side. SVC lack clear ways to interpret the results, while on the contrary RFC allows to have a complete understanding of how the algorithm worked. In any case, to get an idea of which features had greater importance in making the predictions, we report the permutation feature importances in a bar plot.

```
# Evaluate Permutation Feature Importances
f2_scorer = make_scorer(fbeta_score, pos_label=1, beta=2)
importances = pd.DataFrame()
for clf in fitted_models_binary:
    result = permutation_importance(clf, X_train,y_train['Target'],
                                    scoring=f2_scorer,random_state=0)
    result_mean = pd.Series(data=result.importances_mean, index=X.columns)
    importances = pd.concat(objs=[importances,result_mean],axis=1)
importances.columns = clf_str

# Barplot of Feature Importances
fig, axs = plt.subplots(ncols=3, figsize=(20,4))
fig.suptitle('Permutation Feature Importances')
for j, name in enumerate(importances.columns):
    sns.barplot(ax=axs[j], x=importances.index, y=importances[name].values)
    axs[j].tick_params('x',labelrotation=90)
    axs[j].set_ylabel('Importances')
    axs[j].title.set_text(str(name))
plt.show()
```



Remarks on Feature importances:

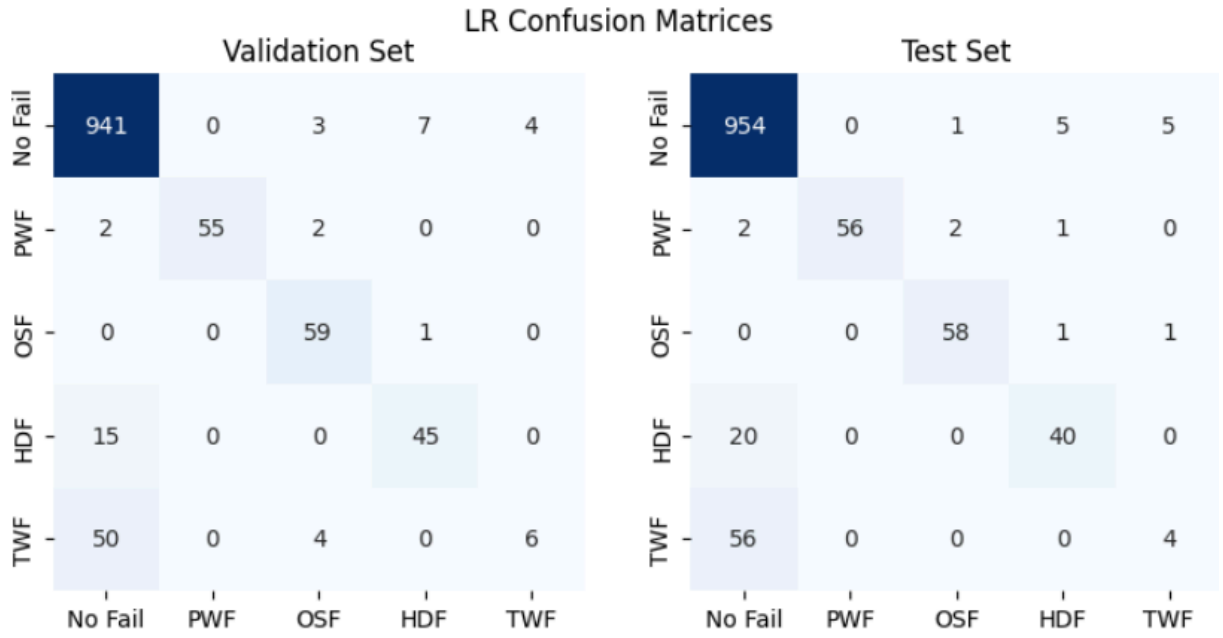
- Type is the feature with the lowest significance, in accordance with what was observed during the exploratory analysis. However, its importance remains strictly positive in each of the cases considered and therefore removing it completely would have led to a decline in prediction performance, not justified by a significant computational gain;
- Unlike Logistic Regression, the models tested place great emphasis on Tool wear as well as Torque and Rotational Speed. Since the former alone is related to a specific category of failures and strongly distorts the kdeplot of Machine failure, we have a sign that our models still worked well.

4. Multi-class task

4.1. Logistic Regression Benchmark

```
# multiclass classification
lr = LogisticRegression(random_state=0,multi_class='ovr')
lr.fit(X_train, y_train['Failure Type'])
y_val_lr = lr.predict(X_val)
y_test_lr = lr.predict(X_test)

# Validation metrics
cm_val_lr, metrics_val_lr = eval_preds(lr,X_val,y_val,y_val_lr,'multi_class')
cm_test_lr, metrics_test_lr = eval_preds(lr,X_test,y_test,y_test_lr,'multi_class')
print('Validation set metrics:',metrics_val_lr, sep='\n')
print('Test set metrics:',metrics_test_lr, sep='\n')

cm_lr = [cm_val_lr, cm_test_lr]
cm_labels = ['No Fail','PWF','OSF','HDF','TWF']
# Show Confusion Matrices
fig, axs = plt.subplots(ncols=2, figsize=(9,4))
fig.suptitle('LR Confusion Matrices')
for j, title in enumerate(['Validation Set', 'Test Set']):
    ax = axs[j]
    sns.heatmap(ax=ax, data=cm_lr[j], annot=True,
                fmt='d', cmap='Blues', cbar=False)
    axs[j].title.set_text(title)
    axs[j].set_xticklabels(cm_labels)
    axs[j].set_yticklabels(cm_labels)
plt.show()

# Odds for interpretation
odds_df = pd.DataFrame(data = np.exp(lr.coef_), columns = X_train.columns,
                       index = df_res['Failure Type'].unique())
odds_df
```

```
Validation set metrics:
ACC     0.926
AUC     0.983
F1      0.909
F2      0.919
dtype: float64
Test set metrics:
ACC     0.922
AUC     0.982
F1      0.904
F2      0.914
dtype: float64
```



In the table above there are, for every class, the Logistic Regression's odds that explain the contribution of each feature in the prediction of belonging to a specific class. By comparing this table with the PCA scatter and the comments we made, we understand that there is a complete agreement about the features that most affect the type of failure. For example, if we look at odds' values of PWF, we see that Rotational Speed and Torque are the ones that are most important for the forecast of belonging to this class. In the analysis of the PCA we stated that PWF seems to be dependent only on PC2, i.e. the Power that is the product of Rotational Speed and Torque. We can make similar considerations for other classes.

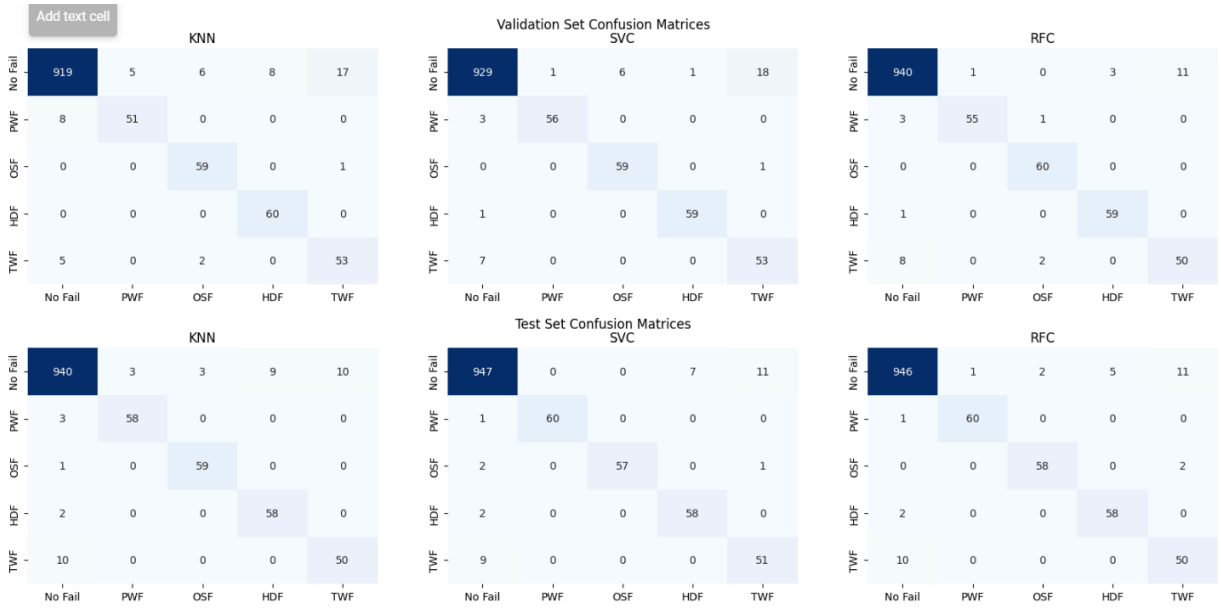
4.2. Models

For each model we launch the Gridsearch for hyperparameter optimization, using as metric to evaluate the model the weighted average F2 score. Similarly to the binary case, the Gridsearch has been started on the parameters that, looking in the literature,

are found to be preponderant for each specific model and the grid values to look for have been defined according to the literature and several tests carried out.

```
GridSearch start
Training KNN
Best params: {'n_neighbors': 1}
Training time: 0m 4s
Training SVC
Best params: {'C': 100, 'gamma': 1, 'kernel': 'rbf', 'probability': True, 'random_state': 0}
Training time: 2m 15s
Training RFC
Best params: {'max_depth': 10, 'n_estimators': 500, 'random_state': 0}
Training time: 3m 37s
```

```
task='multi_class'
y_pred_val, cm_dict_val, metrics_val = predict_and_evaluate(
    fitted_models_multi,X_val,y_val,clf_str,task)
y_pred_test, cm_dict_test, metrics_test = predict_and_evaluate(
    fitted_models_multi,X_test,y_test,clf_str,task)

# Show Validation Confusion Matrices
fig, axs = plt.subplots(ncols=3, figsize=(20,4))
fig.suptitle('Validation Set Confusion Matrices')
for j, model_name in enumerate(clf_str):
    ax = axs[j]
    sns.heatmap(ax=ax, data=cm_dict_val[model_name], annot=True,
                fmt='d', cmap='Blues', cbar=False)
    ax.title.set_text(model_name)
    ax.set_xticklabels(cm_labels)
    ax.set_yticklabels(cm_labels)
plt.show()

# Show Test Confusion Matrices
fig, axs = plt.subplots(ncols=3, figsize=(20,4))
fig.suptitle('Test Set Confusion Matrices')
for j, model_name in enumerate(clf_str):
    ax = axs[j]
    sns.heatmap(ax=ax, data=cm_dict_test[model_name], annot=True,
                fmt='d', cmap='Blues', cbar=False)
    ax.title.set_text(model_name)
    ax.set_xticklabels(cm_labels)
    ax.set_yticklabels(cm_labels)
plt.show()

# Print scores
print('')
print('Validation scores:', metrics_val, sep='\n')
print('Test scores:', metrics_test, sep='\n')
```



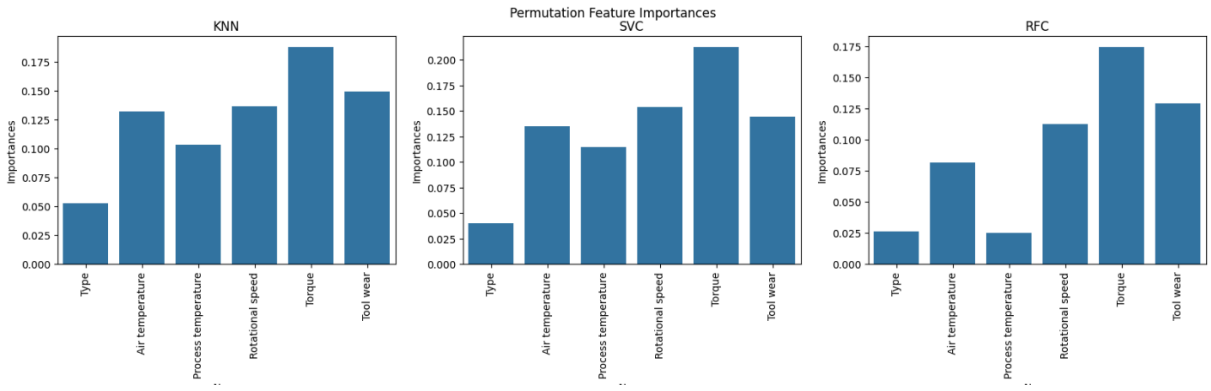

```
Validation scores:
       KNN    SVC    RFC
ACC  0.956  0.968  0.975
AUC  0.956  0.993  0.998
F1   0.957  0.969  0.975
F2   0.957  0.968  0.975
Test scores:
       KNN    SVC    RFC
ACC  0.966  0.973  0.972
AUC  0.956  0.995  0.997
F1   0.966  0.973  0.972
F2   0.966  0.973  0.972
```

By comparing the results obtained, we see that K-NN is the model that performs the worst and its accuracy is a little lower than Logistic Regression's one. Despite this, we cannot exclude it a priori, as it still reaches high values for the metrics and, moreover, gives an immediate response. So, we can use it whenever we need to get an idea quickly about the situation and, then apply other models when we have more time.

All other models perform better than the benchmark and they obtain high values for the chosen metrics both for validation and test set. SVC and RFC's performances are very similar each other. If we look at the training phase, SVC and RFC take the same time. While the best parameters for multiclass K-NN and SVC are the same as binary classification, for RFC the Gridsearch for the two types of task returns different parameters. Moreover, in the transition from binary to multiclass problem, the estimated training time remains the same for all models. In order to understand how

features contribute to predictions, let's look at the Permutation Feature Importances for each model.

```
# Evaluate Permutation Feature Importances
f2_scorer = make_scorer(fbeta_score, beta=2, average='weighted')
importances = pd.DataFrame()
for clf in fitted_models_multi:
    result = permutation_importance(clf, X_train,y_train['Failure Type'],
                                    scoring=f2_scorer,random_state=0)
    result_mean = pd.Series(data=result.importances_mean, index=X.columns)
    importances = pd.concat(objs=[importances,result_mean],axis=1)

importances.columns = clf_str

# Barplot of Feature Importances
fig, axs = plt.subplots(ncols=3, figsize=(20,4))
fig.suptitle('Permutation Feature Importances')
for j, name in enumerate(importances.columns):
    sns.barplot(ax=axs[j], x=importances.index, y=importances[name].values)
    axs[j].tick_params('x',labelrotation=90)
    axs[j].set_ylabel('Importances')
    axs[j].title.set_text(str(name))
plt.show()
```



From previous barplots we see that the models give more importance to Torque, Tool wear and Rotational Speed while the Type contribution is very low. This is in accordance with the observations made in the exploration of the dataset in Section 1-2 and it is consistent with the Permutation Feature Importances of binary task. K-NN is the one who gives more importance to Type, but, different from binary case, here we see that for every model the Type contribution is almost zero. So, we test the model on a new dataset, the old one from which we removed the column Type. For K-NN and SVC there is an insignificant improvement in the metrics' values, which were already very good. For RFC we do not see any change on metrics' values. Since the

training time for the different models is approximately equal in both cases, we let users choose which dataset to use.

5. Conclusions

According to the analyses carried out and the results obtained, it is possible to make some conclusive considerations related to this project.

We decided to tackle two tasks: predict whether a machine will fail or not and predict the type of failure that will occur. Before developing the models we did data preprocessing to ensure the validity of the assumptions of applicability of the models and ensure the best performances. Briefly, in preprocessing phase we have deleted some ambiguous samples, we applied a label encoding to the categorical columns and then we performed the scaling of the columns with StandardScaler. We also noticed the presence of some data points which at first we referred as outliers but later turned out to be part of the natural variance of the data and played an important role in the classification task. Then we ran PCA and found that most of the variance is explained by the first three components, that can be represented as the following features: combination of the two Temperatures, Machine Power (product of Rotational Speed and Torque) and Tool Wear. In according to this, we found that these are the features that contribute the most in the predictions when apply the models. Contrary to logical predictions, we demonstrated that the machine's type does not affect the presence of failure.

At the end, we can conclude that for both task the chosen models perform very well. For both tasks the best model is RF and the worst is KNN; however the response time of KNN is instant while RF takes more time and this further increase when we proceed with the multi-class classification task. The choice of the model depends on the needs of the company: for faster application one can use KNN while if one cares more about accuracy one can use RF.