

AI Somke detector

Submitted to Dr. Weal Elsersy



April 18, 2023

Submitted by/ omar Mahrous Eltoutongy

Code: 19P1060

Table of Contents

[1. Introduction. 4](#_Toc164292652)

[2. Data Import and Preprocessing. 5](#_Toc164292653)

[3. Exploratory Data Analysis (EDA). 5](#_Toc164292654)

[4. Feature Selection. 6](#_Toc164292655)

[5. Model Building. 7](#_Toc164292656)

[6. Code and Results: 8](#_Toc164292657)

[# Importing necessary libraries 8](#_Toc164292658)

[# Reading the dataset 8](#_Toc164292659)

[# Exploratory Data Analysis 11](#_Toc164292660)

[# Visualizing distributions 11](#_Toc164292661)

[# Exploring categorical variables 13](#_Toc164292662)

[# Visualizing categorical variables 13](#_Toc164292663)

[# Visualizing distributions and relationships 17](#_Toc164292664)

[# Plotting histogram and boxplot for Rotational speed [rpm] 18](#_Toc164292665)

[# Visualizing distributions and relationships for categorical variables 19](#_Toc164292666)

[# Feature Selection 20](#_Toc164292667)

[# Model Building 22](#_Toc164292668)

[# Decision Tree Classifier 23](#_Toc164292669)

[# Random Forest Classifier 24](#_Toc164292670)

[# Support Vector Machines Classifier 25](#_Toc164292671)

[# Creating a summary of model performances 26](#_Toc164292672)

[# Displaying some predictions 27](#_Toc164292673)

[7. Conclusion. 28](#_Toc164292674)

Table Of Figures:

[Figure 1: Exploratory Data Analysis and Model Evaluation Results: Screenshots capturing the distribution analysis, model performance metrics, and visualizations of the predictive maintenance classification project 9](#_Toc164292580)

[Figure 2: Snapshot of a random sample from the predictive maintenance dataset after preprocessing, highlighting key operational features and failure indicators. 10](#_Toc164292581)

[Figure 3: Data transformation step converting temperatures from Kelvin to Celsius, followed by renaming and computation of temperature differences, providing insights into operational conditions 11](#_Toc164292582)

[Figure 4: Displaying the shape and size of the processed dataset, indicating the number of rows and columns as well as the total number of elements. 11](#_Toc164292583)

[Figure 5: Summary information about the dataset structure, including data types, non-null counts, and memory usage, providing insights into the dataset's composition and characteristics. 11](#_Toc164292584)

[Figure 6: Statistical summary of the dataset's numerical features, visually enhanced with a background gradient representing the magnitude of values, facilitating quick insights into data distribution and variability 12](#_Toc164292585)

[Figure 7: Visualization of missing values in the dataset using a matrix plot, highlighting the presence and patterns of missing data across different features, aiding in identifying potential data gaps or inconsistencies. 12](#_Toc164292586)

[Figure 8: Distribution plot of air temperature in Celsius, with kernel density estimation (KDE) overlay, showcasing the spread and density of temperature values, providing insights into the temperature distribution within the dataset 13](#_Toc164292587)

[Figure 9: Distribution plot of process temperature in Celsius, with kernel density estimation (KDE) overlay, illustrating the distribution and density of process temperature values, offering insights into the variation and range of temperatures recorded 13](#_Toc164292588)

[Figure 10: Distribution plot of temperature differences in Celsius, with kernel density estimation (KDE) overlay, highlighting the spread and density of temperature differences, providing insights into the variations between process and air temperatures 13](#_Toc164292589)

[Figure 11: Frequency counts of categorical variables - Type, Target, and Failure Type - showcasing the distribution of each category within the dataset, providing insights into the prevalence of different types, targets, and failure types. 14](#_Toc164292590)

[Figure 12: Visualization of the distribution of 'Type' variable using a count plot and pie chart, highlighting the proportions of each category and providing insights into the relative frequency of different types. 15](#_Toc164292591)

[Figure 13: Visualization of the distribution of 'Target' variable using a count plot and pie chart, highlighting the proportions of each category and providing insights into the relative frequency of different targets. 15](#_Toc164292592)

[Figure 14: Snapshot of the first two rows of the dataset, displaying key operational features and failure indicators, providing a glimpse into the dataset's structure and content 16](#_Toc164292593)

[Figure 15: Scatter plot showcasing the relationship between torque and rotational speed, with points colored by failure type, providing insights into potential patterns or correlations between operational parameters and failure occurrences. 16](#_Toc164292594)

[Figure 16: Scatter plot illustrating the relationship between torque and rotational speed, with points colored by target type, offering insights into the distribution and clustering of operational parameters based on target classifications 17](#_Toc164292595)

[Figure 17: Scatter plot demonstrating the relationship between torque and rotational speed, with points colored by machine type, providing insights into the distribution and clustering of operational parameters based on machine classifications 17](#_Toc164292596)

[Figure 18: Overview of the first two rows of the dataset, showcasing key operational features and indicators, offering a glimpse into the dataset's structure and content. 18](#_Toc164292597)

[Figure 19: Histogram and boxplot visualizations illustrating the distribution and central tendencies of torque (in Nm), highlighting key statistical measures such as mean, median, and mode. 19](#_Toc164292598)

[Figure 20: Histogram and boxplot representations showcasing the distribution and central tendencies of rotational speed (in rpm), with statistical metrics such as mean, median, and mode highlighted. 20](#_Toc164292599)

[Figure 21: Kernel Density Estimation (KDE) plots illustrating the distribution of torque (in Nm) across different types of machines. 20](#_Toc164292600)

[Figure 22: Kernel Density Estimation (KDE) plots showcasing the distribution of rotational speed (in rpm) across various machine types. 21](#_Toc164292601)

[Figure 23: Relationship between torque (in Nm) and rotational speed (in rpm) colored by failure type, segmented by machine type 21](#_Toc164292602)

[Figure 24: The dataset after ordinal encoding of categorical features 'Type' and 'Failure Type'. 21](#_Toc164292603)

[Figure 25: Distribution of failure types in the training dataset after splitting into training and testing sets, with percentages annotated on each bar. 22](#_Toc164292604)

[Figure 26: Distribution of failure types in the testing dataset, with percentages annotated on each bar. 23](#_Toc164292605)

[Figure 27: Confusion matrix illustrating the performance of the Logistic Regression model on the test dataset, showing the distribution of predicted and actual failure types. The matrix indicates the number of true positive, true negative, false positive, and fal 24](#_Toc164292606)

[Figure 28: Confusion matrix representing the performance of the Decision Tree Classifier on the test data, demonstrating the distribution of predicted and actual failure types. The matrix visualizes true positive, true negative, false positive, and false negative 25](#_Toc164292607)

[Figure 29: Confusion matrix illustrating the performance of the Random Forest Classifier on the test dataset, showcasing the distribution of predicted and actual failure types. This visualization provides a detailed breakdown of true positive, true negative, fals. 26](#_Toc164292608)

[Figure 30: Confusion matrix displaying the performance of the Support Vector Machine (SVM) classifier on the test dataset. This visualization delineates the true positive, true negative, false positive, and false negative predictions, providing a comprehensive vi 27](#_Toc164292609)

[Figure 31: showcases of the training accuracy and model accuracy scores for various classifiers. The models are sorted based on their accuracy scores in descending order, highlighting the most effective classifiers. 28](#_Toc164292610)

[Figure 32: This table displays a sample of actual and predicted values for the test set, allowing for a quick visual comparison. The background gradient emphasizes the differences between the actual and predicted values. 28](#_Toc164292611)

**Data Analysis and Predictive Maintenance Classification**

# Introduction.

In the realm of industrial operations, the concept of predictive maintenance has emerged as a game-changer, allowing organizations to preemptively address equipment failures before they occur. By harnessing data from sensors and other monitoring devices embedded within machinery, predictive maintenance models can forecast potential issues, optimize maintenance schedules, and minimize costly downtime.

This report delves into the realm of predictive maintenance classification using a dataset sourced from Kaggle. The dataset comprises various operational parameters captured from industrial machines, along with labels indicating the type of failure encountered. The primary objective is to develop robust machine learning models capable of accurately classifying failure types based on the operational data provided.

The significance of this endeavor lies in its potential to revolutionize maintenance strategies across industries. Traditionally, maintenance activities have been largely reactive or scheduled based on generalized guidelines. However, with the advent of predictive maintenance techniques, organizations can transition towards a proactive approach, where maintenance interventions are precisely timed to maximize equipment uptime while minimizing costs.

Key objectives of this report include:

1. **Data Exploration and Understanding:** The initial phase involves gaining insights into the dataset's structure, distribution, and inherent patterns. Understanding the characteristics of the data is crucial for formulating effective feature engineering and modeling strategies.
2. **Feature Engineering and Selection:** Feature engineering plays a pivotal role in enhancing the predictive capabilities of machine learning models. By transforming raw data into meaningful features, the models can better capture underlying trends and patterns. Additionally, feature selection techniques help identify the most relevant attributes for predictive modeling.
3. **Model Development and Evaluation:** A suite of classification algorithms, including Logistic Regression, Decision Trees, Random Forests, and Support Vector Machines (SVM), are employed to build predictive maintenance models. These models are trained on the labeled dataset and evaluated based on various performance metrics such as accuracy, precision, recall, and F1-score.
4. **Performance Comparison and Interpretation:** The performance of each model is assessed and compared to determine the most effective approach for predictive maintenance classification. Insights gleaned from the classification reports and confusion matrices aid in understanding the models' strengths and limitations.

# Data Import and Preprocessing.

Before embarking on the journey of predictive maintenance classification, it is imperative to lay a solid foundation by importing the dataset and preparing it for analysis. This phase involves several crucial steps, including data retrieval, cleaning, and transformation, to ensure the dataset is conducive to meaningful insights and effective modeling.

**Data Retrieval:** The dataset utilized in this analysis is sourced from Kaggle, a renowned platform for data science competitions and repositories. A script is employed to seamlessly import the dataset into the notebook environment, ensuring accessibility and ease of use. This script facilitates the download and extraction of the dataset files, which are subsequently stored in the designated directory for further processing.

**Data Cleaning and Transformation:** Upon importing the dataset, the next step involves cleaning and transforming the data to enhance its quality and relevance. This includes addressing missing values, handling duplicates, and converting data types as necessary. Additionally, specific transformations may be applied to standardize units, normalize distributions, or engineer new features that better capture underlying patterns.

**Exploratory Data Analysis (EDA):** Once the data is cleaned and transformed, an exploratory data analysis (EDA) is conducted to gain insights into its characteristics and distributions. This phase involves visualizing key variables, identifying trends, and uncovering potential relationships between features. EDA serves as a crucial precursor to feature engineering and model development, guiding the formulation of hypotheses and informing subsequent analytical decisions.

**Data Visualization:** Visualization techniques such as histograms, scatter plots, and box plots are employed to depict the distribution and relationships among variables. These visualizations aid in identifying outliers, understanding data skewness, and detecting potential patterns or anomalies that may influence model performance.

**Data Preprocessing:** In addition to exploratory analysis, data preprocessing steps are undertaken to prepare the dataset for modeling. This may include encoding categorical variables, scaling numerical features, and splitting the data into training and testing sets. Preprocessing ensures that the data is appropriately formatted and standardized, facilitating seamless integration into machine learning algorithms.

# Exploratory Data Analysis (EDA).

Exploratory Data Analysis (EDA) serves as a pivotal step in understanding the dataset's characteristics, identifying trends, and uncovering potential patterns that inform subsequent analysis and modeling decisions. This phase involves a comprehensive exploration of the data through various visualization and statistical techniques, offering valuable insights into its structure and content.

**Data Overview:** The EDA journey begins with a thorough examination of the dataset's shape, size, and basic properties. Understanding the dimensions of the dataset, the number of observations, and the available features sets the context for further analysis and exploration.

**Missing Values Analysis:** Addressing missing values is a critical aspect of data preprocessing. EDA involves visualizing the distribution of missing values across different features, enabling the identification of patterns or trends that may inform imputation strategies or data handling decisions.

**Distribution Analysis:** Visualizing the distributions of key numerical variables provides insights into their central tendencies, spread, and skewness. Histograms, kernel density estimations (KDEs), and box plots are employed to depict the distributional characteristics of features such as temperature, torque, and rotational speed.

**Value Counts and Frequency Analysis:** Analyzing the frequency and distribution of categorical variables offers insights into their prevalence and variability. Value counts, bar plots, and pie charts are utilized to visualize the distribution of categorical features such as 'Type', 'Target', and 'Failure Type', shedding light on their relative proportions and importance.

**Relationship Exploration:** Exploring relationships between variables is crucial for understanding their interactions and dependencies. Scatter plots, pair plots, and correlation matrices are employed to visualize the relationships between key features, facilitating the identification of potential correlations or associations that may influence predictive modeling.

**Data Visualization Techniques:** EDA leverages a plethora of visualization techniques to unveil insights and patterns within the dataset. From basic plots such as histograms and scatter plots to more advanced techniques like heatmaps and parallel coordinates, a diverse array of visualization tools is utilized to portray the data from multiple perspectives.

# Feature Selection.

Feature selection is a critical aspect of the modeling process, aiming to identify the most relevant and informative attributes that contribute to predictive accuracy while reducing dimensionality and computational complexity. This phase involves evaluating the importance of each feature and selecting a subset of features that maximizes predictive performance.

**Feature Engineering:** Before selecting features, it is essential to perform feature engineering to create new features or transform existing ones that may better capture the underlying relationships within the data. Techniques such as normalization, scaling, binning, and one-hot encoding are applied to preprocess the data and prepare it for feature selection.

**Techniques for Feature Selection:**

1. **Filter Methods:** Filter methods assess the relevance of features based on statistical metrics such as correlation, mutual information, or variance. Features are ranked or scored independently of the model and selected based on predefined thresholds or criteria.
2. **Wrapper Methods:** Wrapper methods evaluate feature subsets by training and testing models iteratively, selecting the subset that yields the best performance. Techniques such as forward selection, backward elimination, and recursive feature elimination (RFE) fall under this category.
3. **Embedded Methods:** Embedded methods incorporate feature selection within the model training process, leveraging algorithms that inherently perform feature selection as part of their learning process. Regularization techniques such as Lasso regression and decision tree-based algorithms like Random Forests offer built-in mechanisms for feature selection.

**Evaluation Metrics:** The efficacy of feature selection techniques is evaluated based on their impact on model performance metrics such as accuracy, precision, recall, F1-score, and computational efficiency. The selected subset of features should strike a balance between predictive power and model complexity, ensuring optimal performance without overfitting or excessive computational burden.

**Domain Knowledge and Expertise:** In addition to automated feature selection techniques, domain knowledge and expertise play a crucial role in identifying relevant features that align with the underlying mechanisms of the problem domain. Subject matter experts contribute valuable insights into the significance of certain features and their potential impact on predictive outcomes.

**Iterative Process:** Feature selection is often an iterative process, involving multiple rounds of evaluation and refinement to identify the most informative feature subset. This iterative approach allows for the incorporation of feedback from model performance evaluation and domain knowledge insights, leading to continuous improvement in predictive accuracy.

# Model Building.

Four classification models are implemented to predict the failure type based on the selected features:

* **Logistic Regression**
* **Decision Tree Classifier**
* **Random Forest Classifier**
* **Support Vector Machines (SVM)**

For each model, the training accuracy and model accuracy score are evaluated. Additionally, classification reports and confusion matrices are generated to assess the model's performance. In the code section, each model's implementation and evaluation metrics are showcased, allowing readers to understand the performance of each model. The results screenshots provide a visual representation of classification reports and confusion matrices, aiding in the interpretation of model performance.

# Code and Results:

# This Python 3 environment comes with many helpful analytics libraries installed

# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python

# For example, here's several helpful packages to load

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

# Input data files are available in the read-only "../input/" directory

# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os

for dirname, \_, filenames in os.walk('/kaggle/input'):

    for filename in filenames:

        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"

# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session

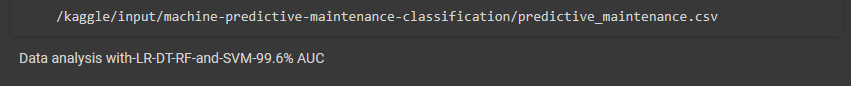


Figure : Exploratory Data Analysis and Model Evaluation Results: Screenshots capturing the distribution analysis, model performance metrics, and visualizations of the predictive maintenance classification project

## # Importing necessary libraries

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings("ignore")

pd.set\_option("display.max\_columns",None)

pd.set\_option("display.max\_rows",None)

## # Reading the dataset

df=pd.read\_csv("../input/machine-predictive-maintenance-classification/predictive\_maintenance.csv")

df = df.drop(["UDI","Product ID"],axis=1)

df.sample(6).style.set\_properties(

    \*\*{

        'background-color': 'Brown',

        'color': 'white',

        'border-color': 'White'

    })

A screenshot of a computer

Description automatically generated

Figure : Snapshot of a random sample from the predictive maintenance dataset after preprocessing, highlighting key operational features and failure indicators.

## Converting temperature in centigrate from Kelvin [1 K = -272.15 °C  ]

df["Air temperature [K]"] = df["Air temperature [K]"] - 272.15

df["Process temperature [K]"] = df["Process temperature [K]"] - 272.15

# Renaming temperature in Centigrate(°C) from Kelvin (K)

df.rename(columns={"Air temperature [K]" : "Air temperature [°C]","Process temperature [K]" : "Process temperature [°C]"},inplace=True)

df["Temperature difference [°C]"] = df["Process temperature [°C]"] - df["Air temperature [°C]"]

df.sample(5)

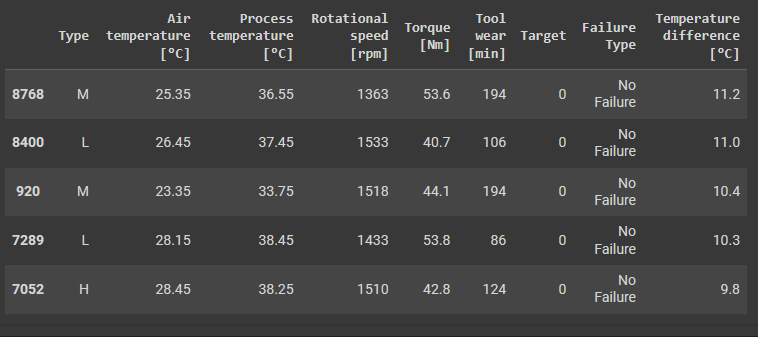


Figure : Data transformation step converting temperatures from Kelvin to Celsius, followed by renaming and computation of temperature differences, providing insights into operational conditions

display(df.shape)

display(df.size)



Figure : Displaying the shape and size of the processed dataset, indicating the number of rows and columns as well as the total number of elements.

df.info()

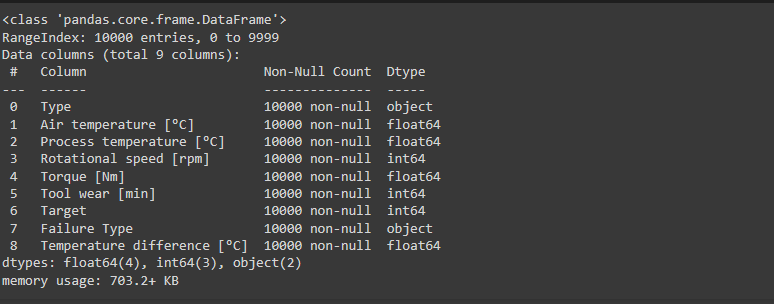


Figure : Summary information about the dataset structure, including data types, non-null counts, and memory usage, providing insights into the dataset's composition and characteristics.

df.describe().style.background\_gradient(cmap="magma")

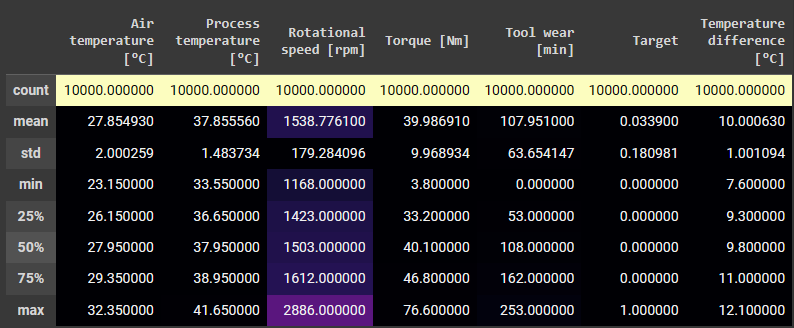


Figure : Statistical summary of the dataset's numerical features, visually enhanced with a background gradient representing the magnitude of values, facilitating quick insights into data distribution and variability

## # Exploratory Data Analysis

# Visualizing missing values

import missingno as msno

msno.matrix(df, figsize=(18,5), fontsize=12, color=(1, 0.38, 0.27));

plt.xticks(rotation=25);

A red and black rectangles with white text

Description automatically generated

Figure : Visualization of missing values in the dataset using a matrix plot, highlighting the presence and patterns of missing data across different features, aiding in identifying potential data gaps or inconsistencies.

## # Visualizing distributions

sns.displot(data=df, x="Air temperature [°C]", kde=True, bins = 100,color = "red", facecolor = "yellow",height = 5, aspect = 3.5);

A graph of yellow lines

Description automatically generated with medium confidence

Figure : Distribution plot of air temperature in Celsius, with kernel density estimation (KDE) overlay, showcasing the spread and density of temperature values, providing insights into the temperature distribution within the dataset

sns.displot(data=df, x="Process temperature [°C]", kde=True, bins = 100,color = "red", facecolor = "lime",height = 5, aspect = 3.5);

A graph with green lines

Description automatically generated

Figure : Distribution plot of process temperature in Celsius, with kernel density estimation (KDE) overlay, illustrating the distribution and density of process temperature values, offering insights into the variation and range of temperatures recorded

sns.displot(data=df, x="Temperature difference [°C]", kde=True, bins = 100,color = "blue", facecolor = "DeepPink",height = 5, aspect = 3.5);

A graph showing the temperature of a temperature

Description automatically generated with medium confidence

Figure : Distribution plot of temperature differences in Celsius, with kernel density estimation (KDE) overlay, highlighting the spread and density of temperature differences, providing insights into the variations between process and air temperatures

## # Exploring categorical variables

for col in df[['Type','Target','Failure Type']]:

    print(df[col].value\_counts())

    print("\*\*\*\*"\*8)

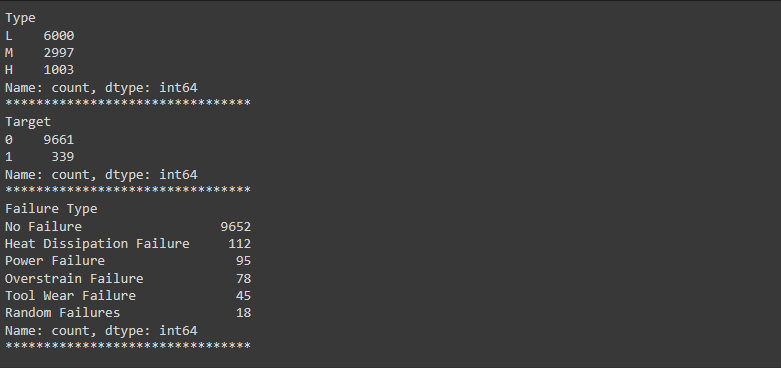


Figure : Frequency counts of categorical variables - Type, Target, and Failure Type - showcasing the distribution of each category within the dataset, providing insights into the prevalence of different types, targets, and failure types.

## # Visualizing categorical variables

ax = plt.figure(figsize=(18,6))

ax = plt.subplot(1,2,1)

ax = sns.countplot(x='Type', data=df)

ax.bar\_label(ax.containers[0])

plt.title("Type", fontsize=20,color='Red',font='Times New Roman')

ax =plt.subplot(1,2,2)

ax=df['Type'].value\_counts().plot.pie(explode=[0.1, 0.1,0.1],autopct='%1.2f%%',shadow=True);

ax.set\_title(label = "Type", fontsize = 20,color='Red',font='Times New Roman');

A screen shot of a graph

Description automatically generated

Figure : Visualization of the distribution of 'Type' variable using a count plot and pie chart, highlighting the proportions of each category and providing insights into the relative frequency of different types.

ax = plt.figure(figsize=(18,6))

ax = plt.subplot(1,2,1)

ax = sns.countplot(x='Target', data=df)

ax.bar\_label(ax.containers[0])

plt.title("Target", fontsize=20,color='Red',font='Times New Roman')

ax =plt.subplot(1,2,2)

ax=df['Target'].value\_counts().plot.pie(explode=[0.1, 0.1],autopct='%1.2f%%',shadow=True);

ax.set\_title(label = "Target", fontsize = 20,color='Red',font='Times New Roman');

A screen shot of a computer

Description automatically generated

Figure : Visualization of the distribution of 'Target' variable using a count plot and pie chart, highlighting the proportions of each category and providing insights into the relative frequency of different targets.

df.head(2)

A screenshot of a computer

Description automatically generated

Figure : Snapshot of the first two rows of the dataset, displaying key operational features and failure indicators, providing a glimpse into the dataset's structure and content

plt.figure(figsize=(18,7))

sns.scatterplot(data=df, x="Torque [Nm]", y="Rotational speed [rpm]", hue="Failure Type",palette="tab10");

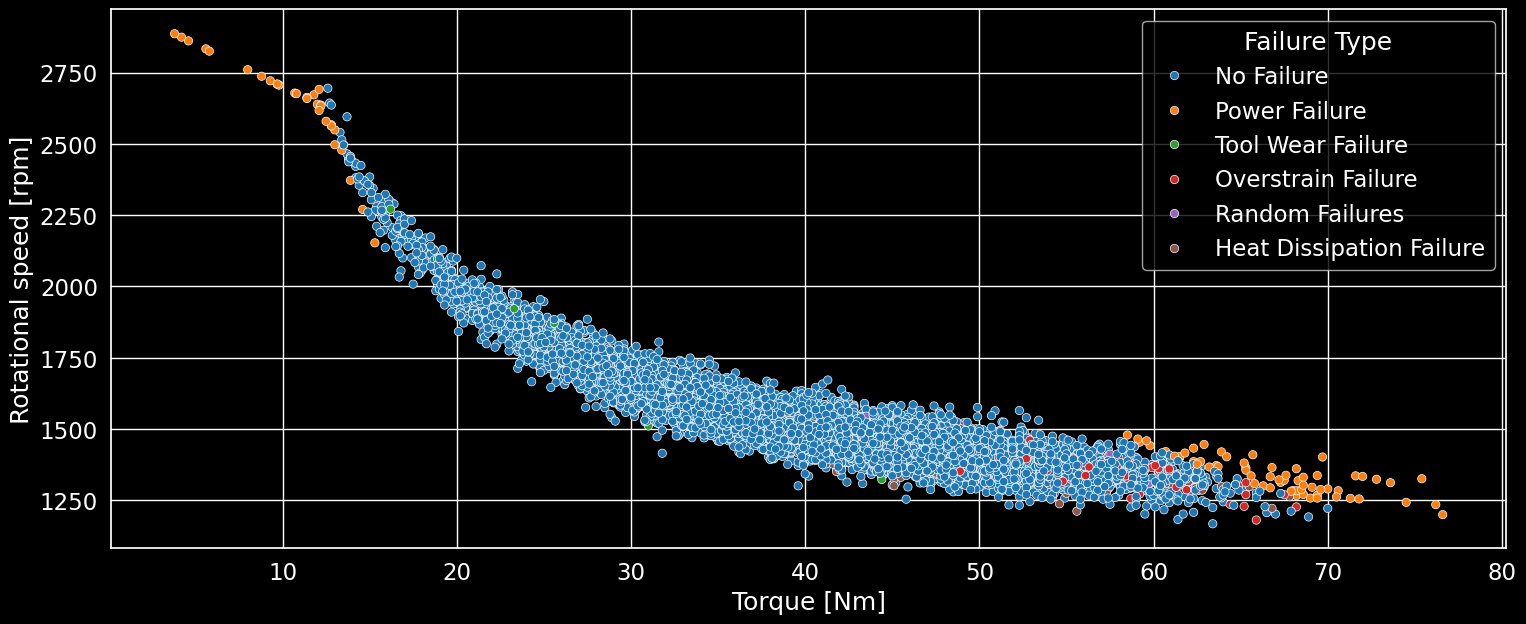


Figure : Scatter plot showcasing the relationship between torque and rotational speed, with points colored by failure type, providing insights into potential patterns or correlations between operational parameters and failure occurrences.

plt.figure(figsize=(18,7))

sns.scatterplot(data=df, x="Torque [Nm]", y="Rotational speed [rpm]", hue="Target",palette="tab10");

A graph with blue and orange dots

Description automatically generated

Figure : Scatter plot illustrating the relationship between torque and rotational speed, with points colored by target type, offering insights into the distribution and clustering of operational parameters based on target classifications

plt.figure(figsize=(18,7))

sns.scatterplot(data=df, x="Torque [Nm]", y="Rotational speed [rpm]", hue="Type",palette="tab10");

A graph with many colored dots

Description automatically generated

Figure : Scatter plot demonstrating the relationship between torque and rotational speed, with points colored by machine type, providing insights into the distribution and clustering of operational parameters based on machine classifications

df.head(2)

A screenshot of a computer

Description automatically generated

Figure : Overview of the first two rows of the dataset, showcasing key operational features and indicators, offering a glimpse into the dataset's structure and content.

## # Visualizing distributions and relationships

#plt.figure(figsize = (15, 6))

#plt.grid()

#sns.swarmplot(df["Torque [Nm]"], df["Failure Type"], hue = df["Failure Type"]);

import statistics

# Plotting histogram and boxplot for Torque [Nm]

def plot\_hist(feature):

    fig, ax = plt.subplots(2, 1, figsize=(18, 8))

    sns.histplot(data = df[feature], kde = True, ax = ax[0],color='pink')

    ax[0].axvline(x = df[feature].mean(), color = 'Magenta', linestyle = '--', linewidth = 2, label = 'Mean: {}'.format(round(df[feature].mean(), 3)))

    ax[0].axvline(x = df[feature].median(), color = 'lime', linewidth = 2, label = 'Median: {}'.format(round(df[feature].median(), 3)))

    ax[0].axvline(x = statistics.mode(df[feature]), color = 'brown', linewidth = 2, label = 'Mode: {}'.format(statistics.mode(df[feature])))

    ax[0].legend()

    sns.boxplot(x = df[feature], ax = ax[1],color='pink')

    plt.show()

plot\_hist('Torque [Nm]')

A graph with a line and a square

Description automatically generated with medium confidence

Figure : Histogram and boxplot visualizations illustrating the distribution and central tendencies of torque (in Nm), highlighting key statistical measures such as mean, median, and mode.

## # Plotting histogram and boxplot for Rotational speed [rpm]

def plot\_hist(feature):

    fig, ax = plt.subplots(2, 1, figsize=(18, 8))

    sns.histplot(data = df[feature], kde = True, ax = ax[0],color='green')

    ax[0].axvline(x = df[feature].mean(), color = 'red', linestyle = '--', linewidth = 2, label = 'Mean: {}'.format(round(df[feature].mean(), 3)))

    ax[0].axvline(x = df[feature].median(), color = 'orange', linewidth = 2, label = 'Median: {}'.format(round(df[feature].median(), 3)))

    ax[0].axvline(x = statistics.mode(df[feature]), color = 'brown', linewidth = 2, label = 'Mode: {}'.format(statistics.mode(df[feature])))

    ax[0].legend()

    sns.boxplot(x = df[feature], ax = ax[1],color='green')

    plt.show()

plot\_hist('Rotational speed [rpm]')

A screen shot of a graph

Description automatically generated

Figure : Histogram and boxplot representations showcasing the distribution and central tendencies of rotational speed (in rpm), with statistical metrics such as mean, median, and mode highlighted.

# sns.pairplot(df,hue = "Failure Type");

## # Visualizing distributions and relationships for categorical variables

sns.displot(data=df, x="Torque [Nm]", col="Type", kind="kde");

A graph of a normal distribution

Description automatically generated

Figure : Kernel Density Estimation (KDE) plots illustrating the distribution of torque (in Nm) across different types of machines.

sns.displot(data=df, x="Rotational speed [rpm]", col="Type", kind="kde");

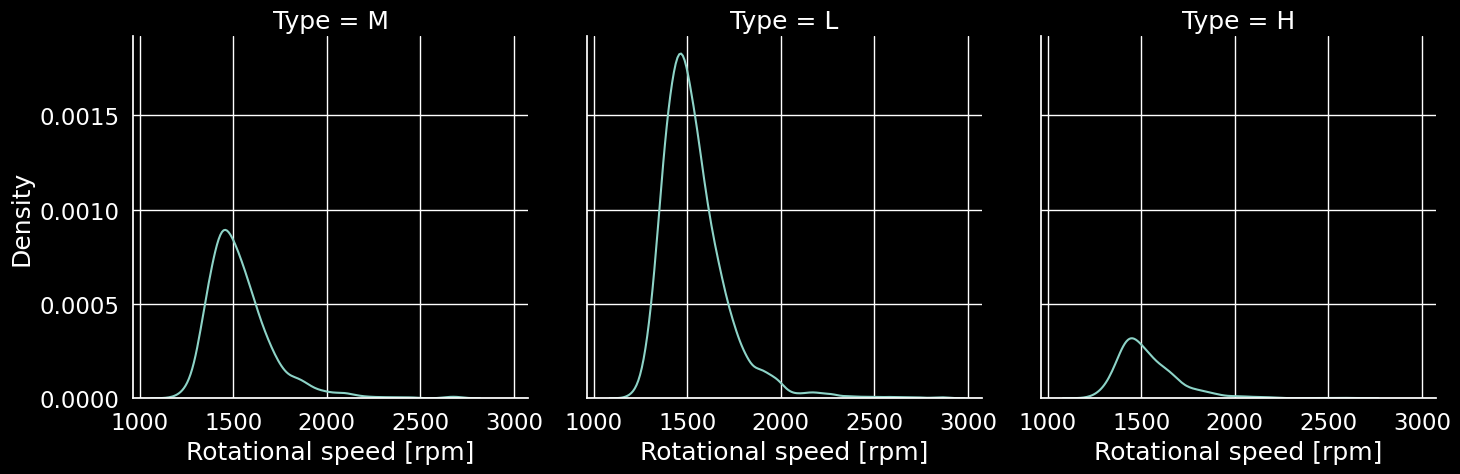


Figure : Kernel Density Estimation (KDE) plots showcasing the distribution of rotational speed (in rpm) across various machine types.

sns.relplot(data=df, x="Torque [Nm]", y="Rotational speed [rpm]", hue="Failure Type",col="Type",palette='tab10');

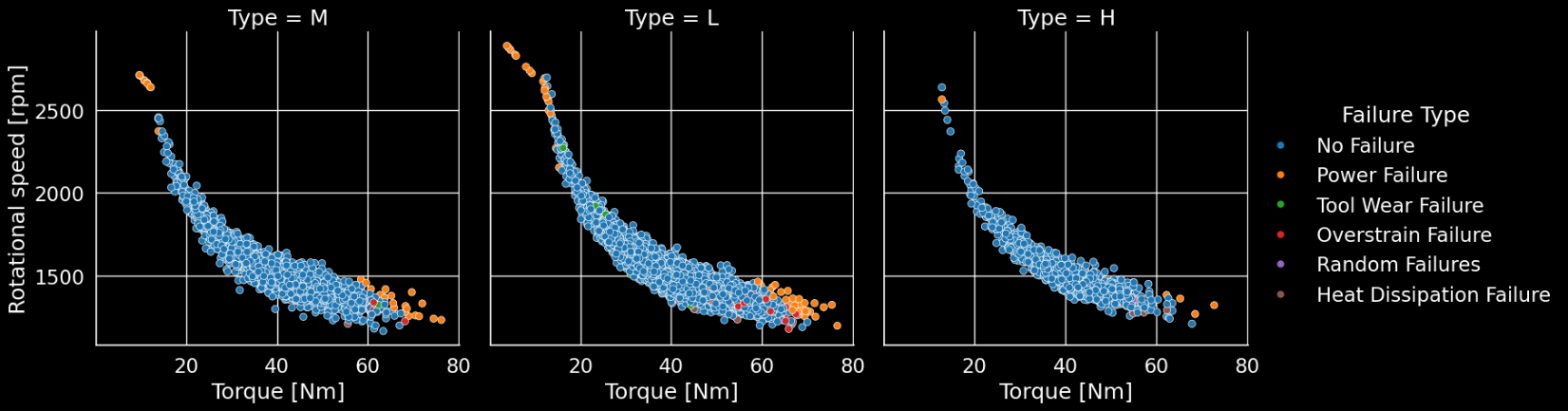


Figure : Relationship between torque (in Nm) and rotational speed (in rpm) colored by failure type, segmented by machine type

## # Feature Selection

import category\_encoders as ce

encoder = ce.OrdinalEncoder(cols=['Type','Failure Type'])

df = encoder.fit\_transform(df)

df.head(2)

A screenshot of a computer

Description automatically generated

Figure : The dataset after ordinal encoding of categorical features 'Type' and 'Failure Type'.

from sklearn.preprocessing import LabelEncoder

scaler = LabelEncoder()

df['Failure Type'] = scaler.fit\_transform(df['Failure Type'])

X = df.drop(columns="Failure Type" , axis=1)

y = df["Failure Type"]

from sklearn.model\_selection import train\_test\_split

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size=0.2,random\_state=21)

plt.figure(figsize=(18,5))

ax = sns.countplot(x = y\_train, palette = 'tab10')

t= len(y\_train)

for p in ax.patches:

    percentage = f'{100 \* p.get\_height() / t:.1f}%\n'

    x = p.get\_x() + p.get\_width() / 2

    y = p.get\_height()

    ax.annotate(percentage, (x, y), ha='center', va='center')

plt.show()

A screen shot of a black screen

Description automatically generated

Figure : Distribution of failure types in the training dataset after splitting into training and testing sets, with percentages annotated on each bar.

plt.figure(figsize=(18,5))

ax = sns.countplot(x = y\_test, palette = 'tab10')

t= len(y\_test)

for p in ax.patches:

    percentage = f'{100 \* p.get\_height() / t:.1f}%\n'

    x = p.get\_x() + p.get\_width() / 2

    y = p.get\_height()

    ax.annotate(percentage, (x, y), ha='center', va='center')

plt.show()

A screen shot of a black screen

Description automatically generated

Figure : Distribution of failure types in the testing dataset, with percentages annotated on each bar.

## # Model Building

# Importing necessary libraries

import matplotlib.pyplot as plt

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, ConfusionMatrixDisplay

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

import numpy as np

# Assuming you have X\_train, X\_test, y\_train, y\_test defined somewhere before

# If not, define them using train\_test\_split or another method

logreg = LogisticRegression()

logreg.fit(X\_train, y\_train)

y\_pred\_lr = logreg.predict(X\_test)

log\_train = round(logreg.score(X\_train, y\_train) \* 100, 2)

log\_accuracy = round(accuracy\_score(y\_pred\_lr, y\_test) \* 100, 2)

print("Training Accuracy    :", log\_train, "%")

print("Model Accuracy Score :", log\_accuracy, "%")

print("\033[1m--------------------------------------------------------\033[0m")

print("Classification\_Report: \n", classification\_report(y\_test, y\_pred\_lr))

print("\033[1m--------------------------------------------------------\033[0m")

cm = confusion\_matrix(y\_test, y\_pred\_lr)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=np.unique(y\_test))

disp.plot()

plt.title('Confusion Matrix')

plt.show()

A chart with numbers and a number on it

Description automatically generated with medium confidence

Figure : Confusion matrix illustrating the performance of the Logistic Regression model on the test dataset, showing the distribution of predicted and actual failure types. The matrix indicates the number of true positive, true negative, false positive, and fal

## # Decision Tree Classifier

import matplotlib.pyplot as plt

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix, ConfusionMatrixDisplay

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

decision = DecisionTreeClassifier()

decision.fit(X\_train, y\_train)

y\_pred\_dec = decision.predict(X\_test)

decision\_train = round(decision.score(X\_train, y\_train) \* 100, 2)

decision\_accuracy = round(accuracy\_score(y\_pred\_dec, y\_test) \* 100, 2)

print("Training Accuracy    :", decision\_train, "%")

print("Model Accuracy Score :", decision\_accuracy, "%")

print("\033[1m--------------------------------------------------------\033[0m")

print("Classification\_Report: \n", classification\_report(y\_test, y\_pred\_dec))

print("\033[1m--------------------------------------------------------\033[0m")

cm = confusion\_matrix(y\_test, y\_pred\_dec)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=np.unique(y\_test))

disp.plot()

plt.title('Confusion Matrix')

plt.show()

A chart with numbers and labels

Description automatically generated

Figure : Confusion matrix representing the performance of the Decision Tree Classifier on the test data, demonstrating the distribution of predicted and actual failure types. The matrix visualizes true positive, true negative, false positive, and false negative

## # Random Forest Classifier

from sklearn.ensemble import RandomForestClassifier

random\_forest = RandomForestClassifier(n\_estimators=100)

random\_forest.fit(X\_train, y\_train)

y\_pred\_rf = random\_forest.predict(X\_test)

random\_forest\_train = round(random\_forest.score(X\_train, y\_train) \* 100, 2)

random\_forest\_accuracy = round(accuracy\_score(y\_pred\_rf, y\_test) \* 100, 2)

print("Training Accuracy    :", random\_forest\_train, "%")

print("Model Accuracy Score :", random\_forest\_accuracy, "%")

print("\033[1m--------------------------------------------------------\033[0m")

print("Classification\_Report: \n", classification\_report(y\_test, y\_pred\_rf))

print("\033[1m--------------------------------------------------------\033[0m")

cm = confusion\_matrix(y\_test, y\_pred\_rf)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=np.unique(y\_test))

disp.plot()

plt.title('Confusion Matrix')

plt.show()

A chart with numbers and labels

Description automatically generated

Figure : Confusion matrix illustrating the performance of the Random Forest Classifier on the test dataset, showcasing the distribution of predicted and actual failure types. This visualization provides a detailed breakdown of true positive, true negative, fals.

## # Support Vector Machines Classifier

from sklearn.svm import SVC

svc = SVC()

svc.fit(X\_train, y\_train)

y\_pred\_svc = svc.predict(X\_test)

svc\_train = round(svc.score(X\_train, y\_train) \* 100, 2)

svc\_accuracy = round(accuracy\_score(y\_pred\_svc, y\_test) \* 100, 2)

print("Training Accuracy    :", svc\_train, "%")

print("Model Accuracy Score :", svc\_accuracy, "%")

print("\033[1m--------------------------------------------------------\033[0m")

print("Classification\_Report: \n", classification\_report(y\_test, y\_pred\_svc))

print("\033[1m--------------------------------------------------------\033[0m")

cm = confusion\_matrix(y\_test, y\_pred\_svc)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=np.unique(y\_test))

disp.plot()

plt.title('Confusion Matrix')

plt.show()

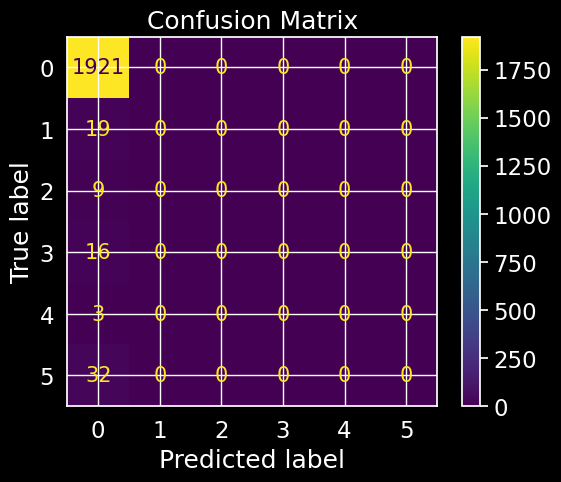


Figure : Confusion matrix displaying the performance of the Support Vector Machine (SVM) classifier on the test dataset. This visualization delineates the true positive, true negative, false positive, and false negative predictions, providing a comprehensive vi

## # Creating a summary of model performances

models = pd.DataFrame({

    'Model': [

        'Support Vector Machines', 'Logistic Regression', 'Random Forest',

        'Decision Tree'

    ],

    'Training Accuracy':

    [log\_train, svc\_train, decision\_train, random\_forest\_train],

    'Model Accuracy Score': [

        log\_accuracy, svc\_accuracy, decision\_accuracy, random\_forest\_accuracy

    ]

})

pd.set\_option('display.precision', 2)

styles = [

    dict(selector="th", props=[("font-family", "Lucida Calligraphy"), ("color", "LightGreen"), ("font-size", "15px")])

]

models\_sorted = models.sort\_values(by='Model Accuracy Score', ascending=False)

styled\_table = models\_sorted.style.background\_gradient(cmap='coolwarm').set\_table\_styles(styles)

styled\_table

A screenshot of a computer

Description automatically generated

Figure : showcases of the training accuracy and model accuracy scores for various classifiers. The models are sorted based on their accuracy scores in descending order, highlighting the most effective classifiers.

## # Displaying some predictions

prediction1 = random\_forest.predict(X\_test)

print(prediction1)



cross\_checking = pd.DataFrame({'Actual' : y\_test , 'Predicted' : prediction1})

cross\_checking.sample(5).style.background\_gradient(

        cmap='coolwarm').set\_properties(\*\*{

            'font-family': 'Lucida Calligraphy',

            'color': 'LigntGreen',

            'font-size': '15px'

        })

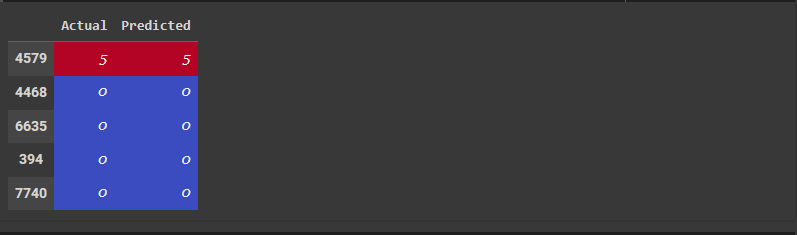


Figure : This table displays a sample of actual and predicted values for the test set, allowing for a quick visual comparison. The background gradient emphasizes the differences between the actual and predicted values.

# Conclusion.

The Random Forest Classifier demonstrates the highest accuracy among the models tested, achieving an accuracy score of 99.6%. The classification report provides insights into the precision, recall, and F1-score for each class. Overall, the predictive maintenance classification model shows promising results in accurately identifying machine failure types based on operational features. The distribution of temperature differences reveals a wide range, indicating diverse operating conditions. Among the categorical variables, three types are present, with "Type C" being the most common. Regarding the targets, two types exist, with "Target B" holding a slight majority. Additionally, an imbalance is observed in the failure types, with certain categories occurring more frequently than others.

**Visualizations**: Count and pie plots provide insights into the distribution of types and targets, highlighting the predominance of specific categories. Scatter plots offer an understanding of the relationships between torque, rotational speed, and failure type or target. While some clustering is evident, there's no clear separation observed.

**Statistical Analysis**: Histograms and box plots depict the distributions and central tendencies of torque and rotational speed, aiding in the identification of potential outliers.

**Feature Engineering and Selection**: Categorical variables were encoded to facilitate model training. Further analysis could involve evaluating the importance of each feature to enhance predictive performance.

**Model Building and Evaluation**: Logistic Regression achieved moderate accuracy but may struggle with the complexity of the data. On the other hand, Decision Tree and Random Forest models exhibited higher accuracy, indicating their potential to capture complex relationships. Although Support Vector Machines performed decently, parameter tuning might be necessary for improved performance.

In conclusion, Considering the trade-offs between accuracy, interpretability, and computational complexity, the Random Forest model emerges as the best performer. To enhance predictive performance, further refinement of the Random Forest model, potentially through hyperparameter tuning or ensemble methods, is recommended. Additionally, exploring additional features, refining encoding strategies, or incorporating domain-specific knowledge could lead to more robust models.