In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings("ignore")
```

In [7]:
```
sns.set_theme(palette='tab10',
              font_scale=1.5,
              rc=None)

import matplotlib
matplotlib.rcParams.update({'font.size': 15})
```

In [8]:
```
df=pd.read_csv("/content/drive/MyDrive/predictive_maintenance.csv")
df = df.drop(["UDI","Product ID"],axis=1)
df.head()
```

Out[8]:

	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Target	Failure Type
0	M	298.1	308.6	1551	42.8	0	0	No Failure
1	L	298.2	308.7	1408	46.3	3	0	No Failure
2	L	298.1	308.5	1498	49.4	5	0	No Failure
3	L	298.2	308.6	1433	39.5	7	0	No Failure
4	L	298.2	308.7	1408	40.0	9	0	No Failure

In [9]:

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

df.rename(columns={"Air temperature [K]" : "Air temperature [C]","Process t
df["Temperature difference [C]"] = df["Process temperature [C]"] - df["Air

df.head()
```

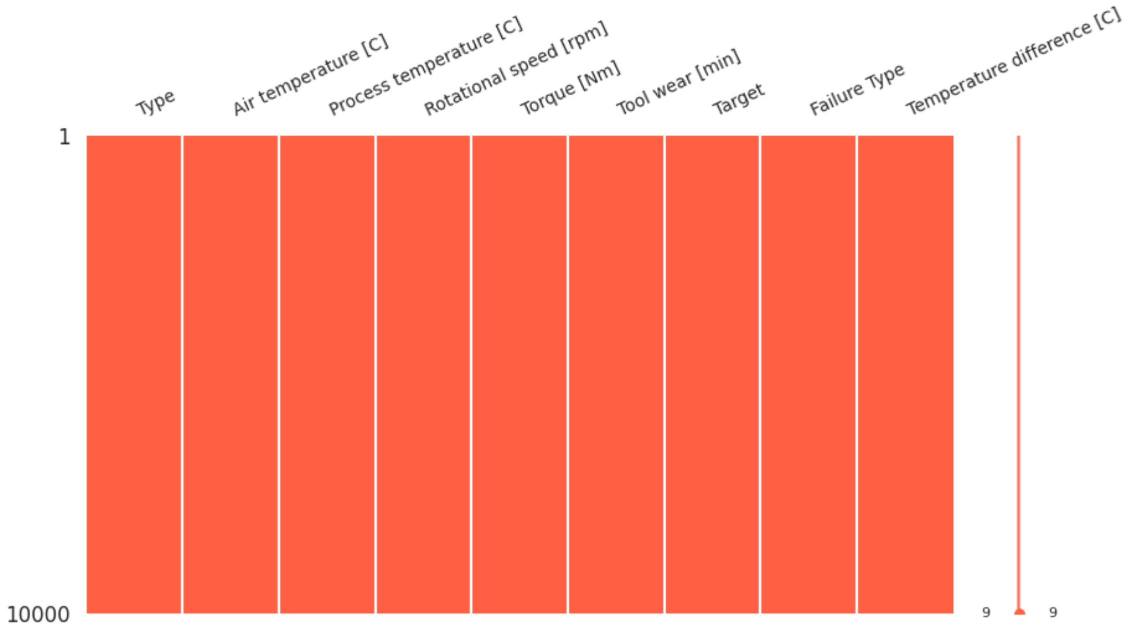
Out[9]:

	Type	Air temperature [C]	Process temperature [C]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Target	Failure Type	Temperature difference [C]
0	M	25.95	36.45	1551	42.8	0	0	No Failure	10.5
1	L	26.05	36.55	1408	46.3	3	0	No Failure	10.5
2	L	25.95	36.35	1498	49.4	5	0	No Failure	10.4
3	L	26.05	36.45	1433	39.5	7	0	No Failure	10.4
4	L	26.05	36.55	1408	40.0	9	0	No Failure	10.5

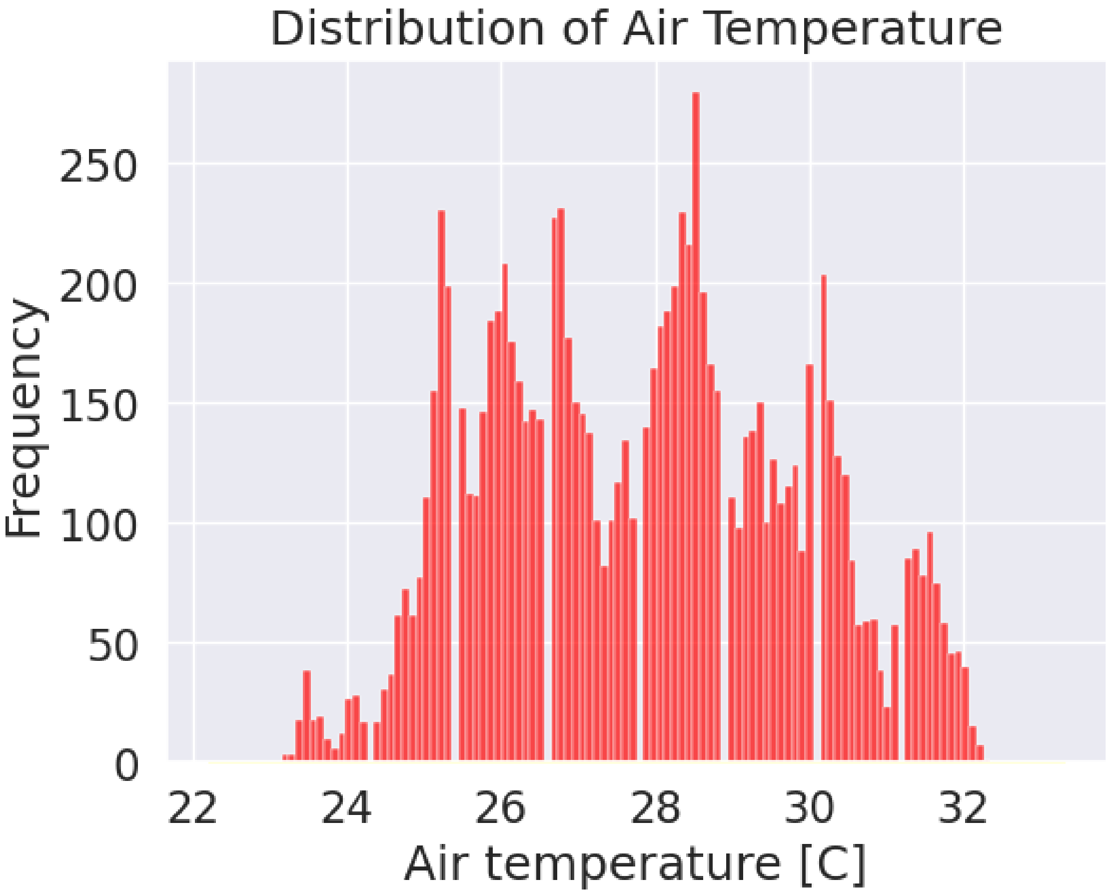
Exploratory Data Analysis

In [10]:
```
import missingno as msno
msno.matrix(df, figsize=(10,5), fontsize=10, color=(1, 0.38, 0.27));
plt.xticks(rotation=25)
```

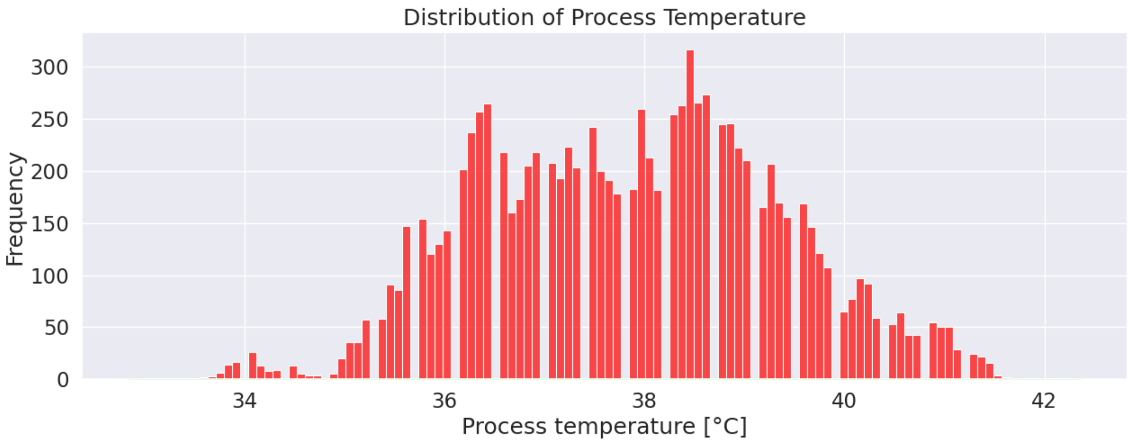
Out[10]:
```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8]),
 [Text(0, 1, 'Type'),
  Text(1, 1, 'Air temperature [C]'),
  Text(2, 1, 'Process temperature [C]'),
  Text(3, 1, 'Rotational speed [rpm]'),
  Text(4, 1, 'Torque [Nm]'),
  Text(5, 1, 'Tool wear [min]'),
  Text(6, 1, 'Target'),
  Text(7, 1, 'Failure Type'),
  Text(8, 1, 'Temperature difference [C]')])
```



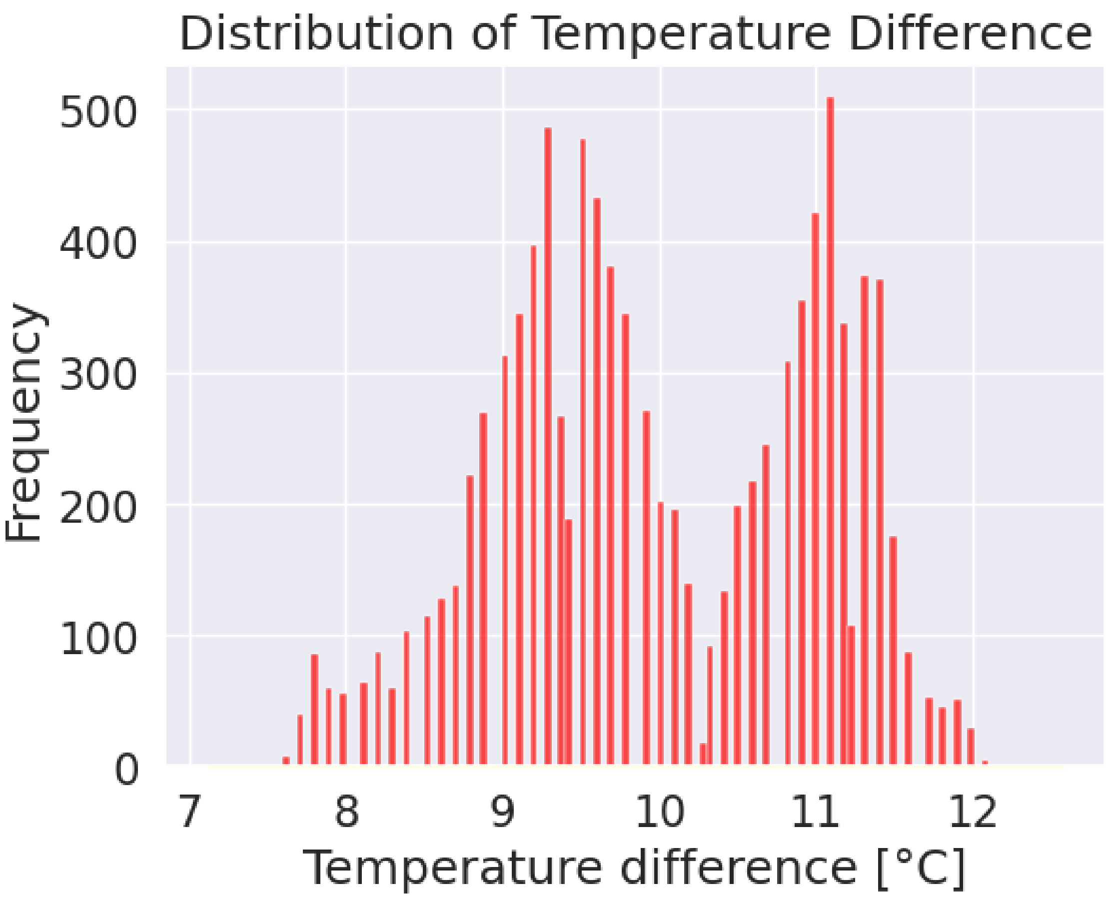
In [11]:
```
sns.histplot(data=df, x="Air temperature [C]", bins=100, color="red", alpha

# Kernel Density Estimate (KDE)
sns.kdeplot(data=df, x="Air temperature [C]", color="yellow", fill=True)

plt.title('Distribution of Air Temperature')
plt.xlabel('Air temperature [C]')
plt.ylabel('Frequency')
plt.show()
```



In [12]:
```
plt.figure(figsize=(15, 5))

# Histogram
sns.histplot(data=df, x="Process temperature [C]", bins=100, color="red", a

# Kernel Density Estimate (KDE)
sns.kdeplot(data=df, x="Process temperature [C]", color="lime", fill=True)

plt.title('Distribution of Process Temperature')
plt.xlabel('Process temperature [°C]')
plt.ylabel('Frequency')
plt.show()
```

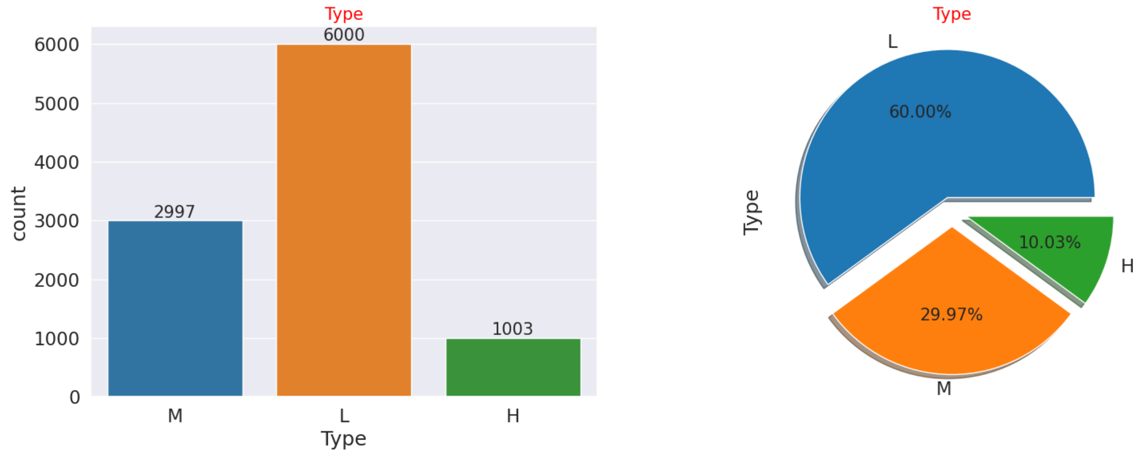


In [13]:
```
sns.histplot(data=df, x="Temperature difference [C]", bins=100, color="red"

# Kernel Density Estimate (KDE)
sns.kdeplot(data=df, x="Temperature difference [C]", color="yellow", fill=T

plt.title('Distribution of Temperature Difference')
plt.xlabel('Temperature difference [°C]')
plt.ylabel('Frequency')
plt.show()
```



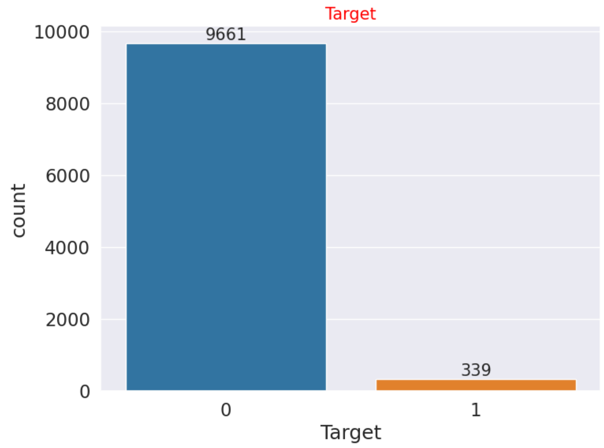
In [14]:
```
# Set up the subplots
fig, axes = plt.subplots(1, 2, figsize=(18, 6))

# Count plot
sns.countplot(x='Type', data=df, ax=axes[0])
axes[0].bar_label(axes[0].containers[0])
axes[0].set_title("Type", fontsize=20, color='Red', font='Times New Roman')

# Pie chart
df['Type'].value_counts().plot.pie(explode=[0.1, 0.1, 0.1], autopct='%1.2f%
axes[1].set_title("Type", fontsize=20, color='Red', font='Times New Roman')

plt.show()
```

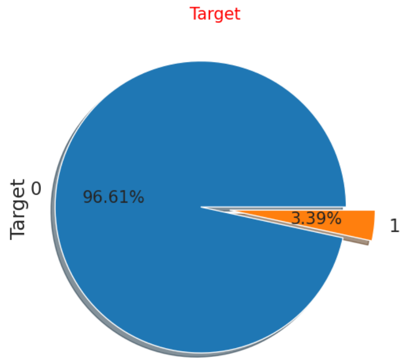
```
WARNING:matplotlib.font_manager:findfont: Font family 'Times New Roman' no
t found.
WARNING:matplotlib.font_manager:findfont: Font family 'Times New Roman' no
t found.
WARNING:matplotlib.font_manager:findfont: Font family 'Times New Roman' no
t found.
WARNING:matplotlib.font_manager:findfont: Font family 'Times New Roman' no
t found.
WARNING:matplotlib.font_manager:findfont: Font family 'Times New Roman' no
t found.
WARNING:matplotlib.font_manager:findfont: Font family 'Times New Roman' no
t found.
```



In [15]:
```
fig, axes = plt.subplots(1, 2, figsize=(18, 6))

# Count plot
sns.countplot(x='Target', data=df, ax=axes[0])
axes[0].bar_label(axes[0].containers[0])
axes[0].set_title("Target", fontsize=20, color='Red', font='Times New Roman

# Pie chart
df['Target'].value_counts().plot.pie(explode=[0.1, 0.1], autopct='%1.2f%%',
axes[1].set_title("Target", fontsize=20, color='Red', font='Times New Roman

plt.show()
```

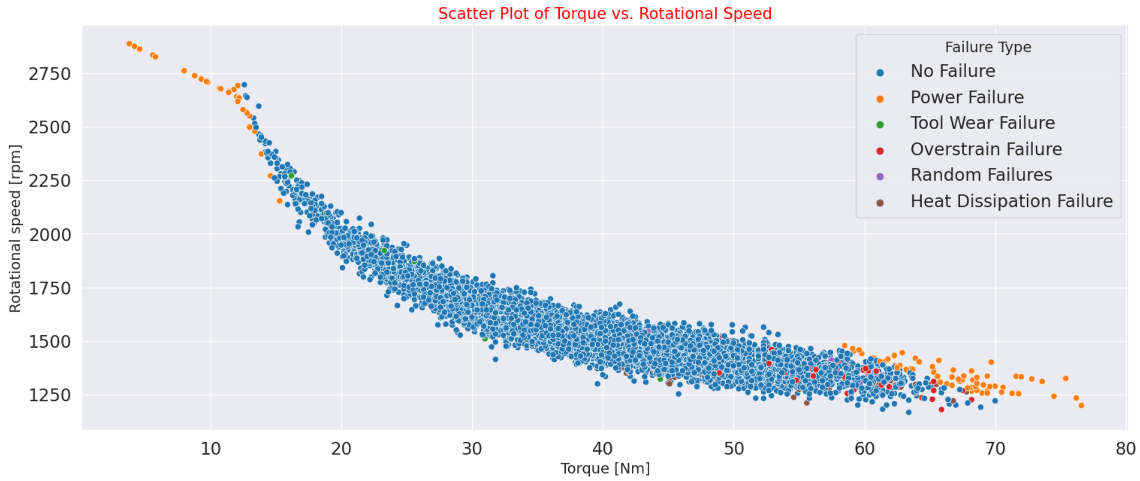
```
WARNING:matplotlib.font_manager:findfont: Font family 'Times New Roman' no
t found.
WARNING:matplotlib.font_manager:findfont: Font family 'Times New Roman' no
t found.
WARNING:matplotlib.font_manager:findfont: Font family 'Times New Roman' no
t found.
WARNING:matplotlib.font_manager:findfont: Font family 'Times New Roman' no
t found.
WARNING:matplotlib.font_manager:findfont: Font family 'Times New Roman' no
t found.
WARNING:matplotlib.font_manager:findfont: Font family 'Times New Roman' no
t found.
```





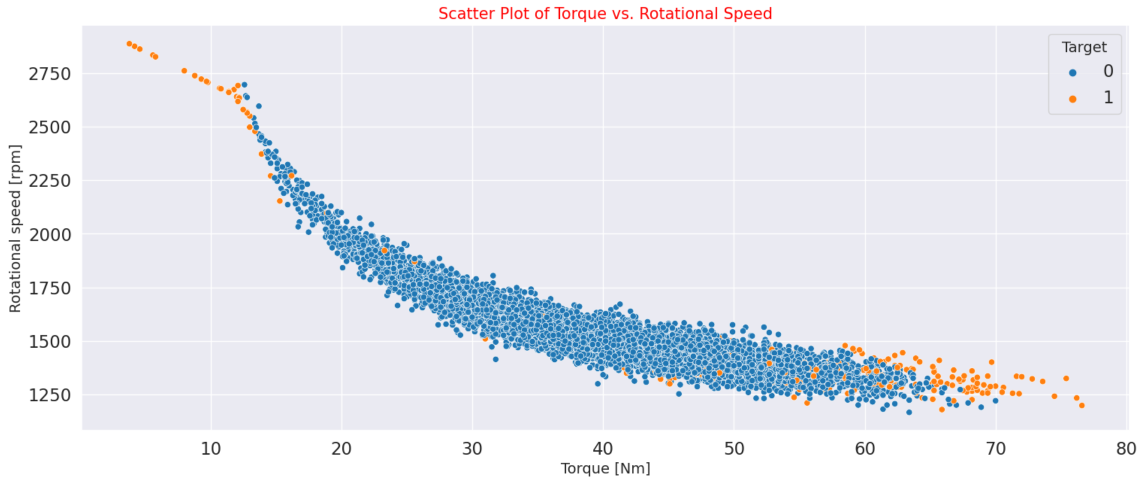
In [16]:
```
plt.figure(figsize=(18, 7))

# Use scatterplot function
sns.scatterplot(data=df, x="Torque [Nm]", y="Rotational speed [rpm]", hue="

plt.title("Scatter Plot of Torque vs. Rotational Speed", fontsize=20, color
plt.xlabel("Torque [Nm]", fontsize=14)
plt.ylabel("Rotational speed [rpm]", fontsize=14)
plt.legend(title="Failure Type", title_fontsize='14', loc='upper right')

plt.show()
```

```
WARNING:matplotlib.font_manager:findfont: Font family 'Times New Roman' no
t found.
WARNING:matplotlib.font_manager:findfont: Font family 'Times New Roman' no
t found.
WARNING:matplotlib.font_manager:findfont: Font family 'Times New Roman' no
t found.
WARNING:matplotlib.font_manager:findfont: Font family 'Times New Roman' no
t found.
WARNING:matplotlib.font_manager:findfont: Font family 'Times New Roman' no
t found.
```



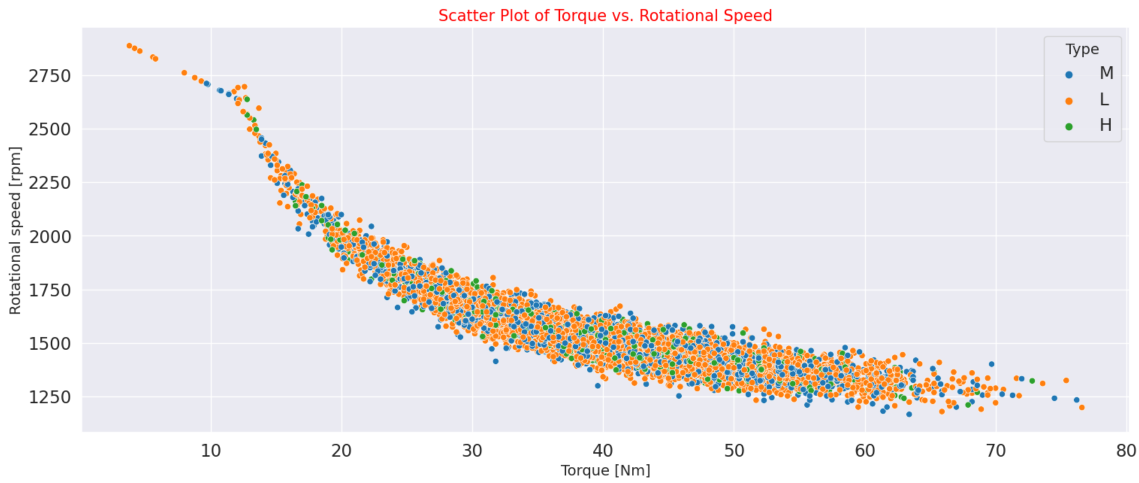
In [17]:
```
# Use scatterplot function
plt.figure(figsize=(18, 7))

sns.scatterplot(data=df, x="Torque [Nm]", y="Rotational speed [rpm]", hue="

plt.title("Scatter Plot of Torque vs. Rotational Speed", fontsize=20, color
plt.xlabel("Torque [Nm]", fontsize=14)
plt.ylabel("Rotational speed [rpm]", fontsize=14)
plt.legend(title="Target", title_fontsize='14', loc='upper right')

plt.show()
```

```
WARNING:matplotlib.font_manager:findfont: Font family 'Times New Roman' no
t found.
WARNING:matplotlib.font_manager:findfont: Font family 'Times New Roman' no
t found.
WARNING:matplotlib.font_manager:findfont: Font family 'Times New Roman' no
t found.
WARNING:matplotlib.font_manager:findfont: Font family 'Times New Roman' no
t found.
WARNING:matplotlib.font_manager:findfont: Font family 'Times New Roman' no
t found.
```

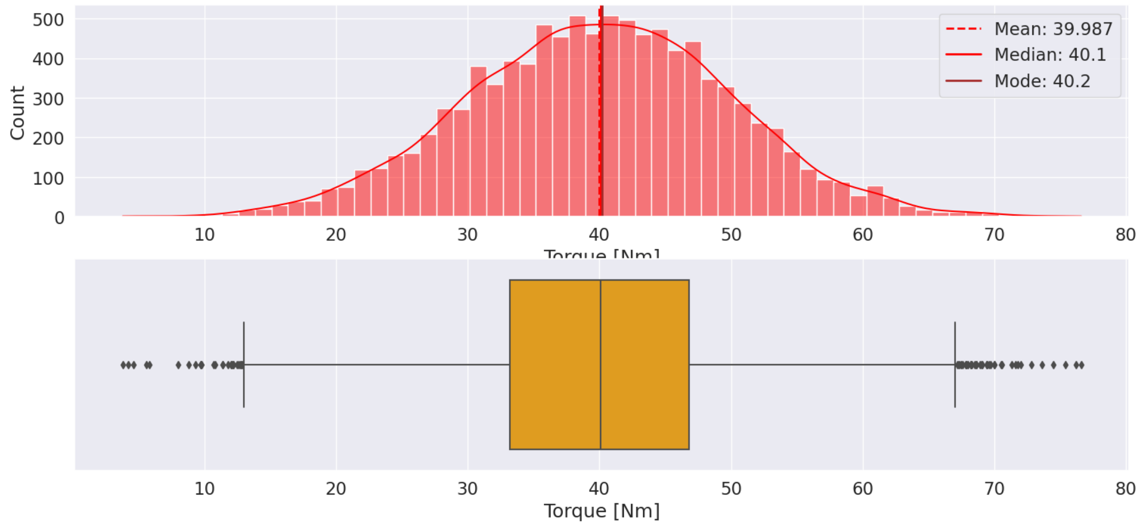


In [18]:
```
plt.figure(figsize=(18, 7))

# Use scatterplot function
sns.scatterplot(data=df, x="Torque [Nm]", y="Rotational speed [rpm]", hue="

plt.title("Scatter Plot of Torque vs. Rotational Speed", fontsize=20, color
plt.xlabel("Torque [Nm]", fontsize=14)
plt.ylabel("Rotational speed [rpm]", fontsize=14)
plt.legend(title="Type", title_fontsize='14', loc='upper right')

plt.show()
```

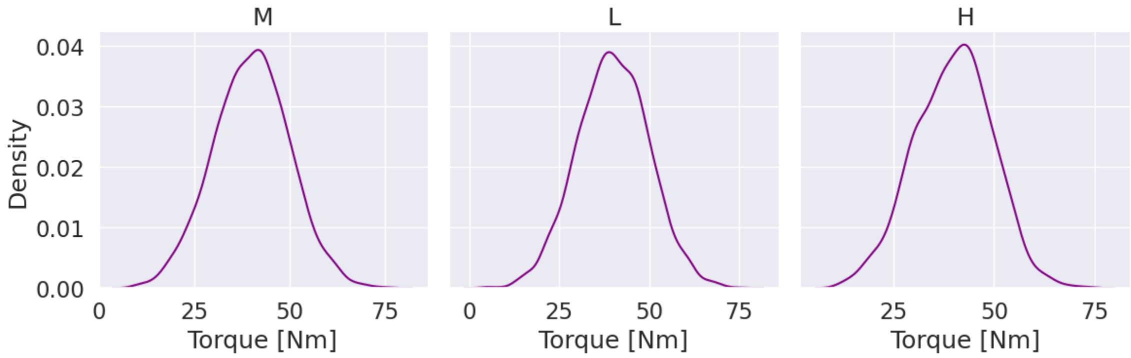
```
WARNING:matplotlib.font_manager:findfont: Font family 'Times New Roman' no
t found.
WARNING:matplotlib.font_manager:findfont: Font family 'Times New Roman' no
t found.
WARNING:matplotlib.font_manager:findfont: Font family 'Times New Roman' no
t found.
WARNING:matplotlib.font_manager:findfont: Font family 'Times New Roman' no
t found.
WARNING:matplotlib.font_manager:findfont: Font family 'Times New Roman' no
t found.
```



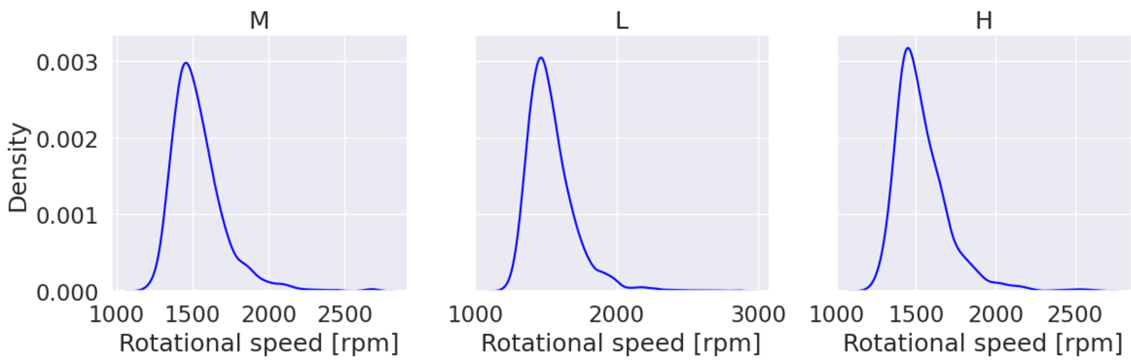
In [19]:
```
import statistics
import os
def plot_hist(feature):
    fig, axes = plt.subplots(2, 1, figsize=(18, 8))

    # Histogram with KDE
    sns.histplot(data=df[feature], kde=True, ax=axes[0], color='red')
    axes[0].axvline(x=df[feature].mean(), color='red', linestyle='--', line
    axes[0].axvline(x=df[feature].median(), color='red', linewidth=2, label
    axes[0].axvline(x=statistics.mode(df[feature]), color='brown', linewidt
    axes[0].legend()

    # Boxplot
    sns.boxplot(x=df[feature], ax=axes[1], color='orange')

    plt.show()

# Example usage
plot_hist('Torque [Nm]')
```



In [20]:
```
# Set up the subplots
g = sns.FacetGrid(df, col="Type", col_wrap=3, height=4, sharex=False)

# Use the map function to plot KDE on each subplot
g.map(sns.kdeplot, "Torque [Nm]", color="purple")

# Set common axis labels
g.set_axis_labels("Torque [Nm]", "Density")

# Add titles to subplots
g.set_titles(col_template="{col_name}")

plt.show()
```



In [21]:
```
# Set up the subplots
g = sns.FacetGrid(df, col="Type", col_wrap=3, height=4, sharex=False)

# Use the map function to plot KDE on each subplot
g.map(sns.kdeplot, "Rotational speed [rpm]", color="blue")

# Set common axis labels
g.set_axis_labels("Rotational speed [rpm]", "Density")

# Add titles to subplots
g.set_titles(col_template="{col_name}")

plt.show()
```



Feature Selection

In [23]:
```
!pip install --upgrade category_encoders
```

```
Collecting category_encoders
  Downloading category_encoders-2.6.3-py2.py3-none-any.whl (81 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 81.9/81.9 kB 988.8 kB/s eta
0:00:00
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python3.10/
dist-packages (from category_encoders) (1.23.5)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/pyth
on3.10/dist-packages (from category_encoders) (1.2.2)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.10/d
ist-packages (from category_encoders) (1.11.3)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python
3.10/dist-packages (from category_encoders) (0.14.0)
Requirement already satisfied: pandas>=1.0.5 in /usr/local/lib/python3.10/
dist-packages (from category_encoders) (1.5.3)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.10/d
ist-packages (from category_encoders) (0.5.3)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/py
thon3.10/dist-packages (from pandas>=1.0.5->category_encoders) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/d
ist-packages (from pandas>=1.0.5->category_encoders) (2023.3.post1)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packa
ges (from patsy>=0.5.1->category_encoders) (1.16.0)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/
dist-packages (from scikit-learn>=0.20.0->category_encoders) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/pyth
on3.10/dist-packages (from scikit-learn>=0.20.0->category_encoders) (3.2.
0)
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.1
0/dist-packages (from statsmodels>=0.9.0->category_encoders) (23.2)
Installing collected packages: category_encoders
Successfully installed category_encoders-2.6.3
```

In [24]:
```
import category_encoders as ce
from sklearn.model_selection import train_test_split

encoder = ce.OrdinalEncoder(cols=['Type','Failure Type'])
df = encoder.fit_transform(df)
X = df.drop(columns="Failure Type" , axis=1)
y = df["Failure Type"]
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_s
```

Logistic Regression

In [42]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
# from sklearn.metrics import plot_confusion_matrix
from sklearn import metrics

log_train=0
log_accuracy=0
def logistic_regression_model(X_train, y_train, X_test, y_test):
    # Create and train the logistic regression model
    logreg = LogisticRegression()
    logreg.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred_lr = logreg.predict(X_test)

    # Calculate training accuracy
    log_train = round(logreg.score(X_train, y_train) * 100, 2)

    # Calculate accuracy score on the test set
    log_accuracy = round(accuracy_score(y_pred_lr, y_test) * 100, 2)

    # Print results
    print("Training Accuracy    :", log_train, "%")
    print("Model Accuracy Score :", log_accuracy, "%")
    print("Classification_Report: \n", classification_report(y_test, y_pred



    plt.show()

# Example usage
logistic_regression_model(X_train, y_train, X_test, y_test)
```

```
Training Accuracy    : 96.79 %
Model Accuracy Score : 96.25 %
Classification_Report: 
               precision    recall  f1-score   support

           1       0.96      1.00      0.98      1921
           2       0.00      0.00      0.00        19
           3       0.00      0.00      0.00         9
           4       0.67      0.38      0.48        16
           5       0.00      0.00      0.00         3
           6       0.00      0.00      0.00        32

    accuracy                           0.96      2000
   macro avg       0.27      0.23      0.24      2000
weighted avg       0.93      0.96      0.95      2000
```

Decision Tree Classifier

In [45]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
decision_train=0
decision_accuracy=0
def decision_tree_model(X_train, y_train, X_test, y_test):
    # Create and train the decision tree model
    decision = DecisionTreeClassifier()
    decision.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred_dec = decision.predict(X_test)

    # Calculate training accuracy
    decision_train = round(decision.score(X_train, y_train) * 100, 2)

    # Calculate accuracy score on the test set
    decision_accuracy = round(accuracy_score(y_pred_dec, y_test) * 100, 2)

    # Print results
    print("Training Accuracy    :", decision_train, "%")
    print("Model Accuracy Score :", decision_accuracy, "%")
    print("Classification_Report: \n", classification_report(y_test, y_pred

    plt.show()

# Example usage
decision_tree_model(X_train, y_train, X_test, y_test)
```

```
Training Accuracy    : 100.0 %
Model Accuracy Score : 99.2 %
Classification_Report: 
               precision    recall  f1-score   support

           1       1.00      1.00      1.00      1921
           2       0.81      0.89      0.85        19
           3       0.90      1.00      0.95         9
           4       0.92      0.75      0.83        16
           5       0.00      0.00      0.00         3
           6       0.97      0.97      0.97        32

    accuracy                           0.99      2000
   macro avg       0.77      0.77      0.77      2000
weighted avg       0.99      0.99      0.99      2000
```

Random Forest Classifier

In [46]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
random_forest_train=0
random_forest_accuracy=0
def random_forest_model(X_train, y_train, X_test, y_test, n_estimators=100)
    # Create and train the Random Forest model
    random_forest = RandomForestClassifier(n_estimators=n_estimators)
    random_forest.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred_rf = random_forest.predict(X_test)

    # Calculate training accuracy
    random_forest_train = round(random_forest.score(X_train, y_train) * 100

    # Calculate accuracy score on the test set
    random_forest_accuracy = round(accuracy_score(y_pred_rf, y_test) * 100,

    # Print results
    print("Training Accuracy    :", random_forest_train, "%")
    print("Model Accuracy Score :", random_forest_accuracy, "%")
    print("Classification_Report: \n", classification_report(y_test, y_pred


    plt.show()

# Example usage
random_forest_model(X_train, y_train, X_test, y_test, n_estimators=100)
```

```
Training Accuracy    : 100.0 %
Model Accuracy Score : 99.6 %
Classification_Report:
               precision    recall  f1-score   support

           1       1.00      1.00      1.00      1921
           2       0.90      0.95      0.92        19
           3       1.00      0.78      0.88         9
           4       0.88      0.94      0.91        16
           5       0.00      0.00      0.00         3
           6       0.97      0.97      0.97        32

    accuracy                           1.00      2000
   macro avg       0.79      0.77      0.78      2000
weighted avg       0.99      1.00      1.00      2000
```

SVM

In [47]:
```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt
svc_train=0
svc_accuracy=0
def support_vector_machine_model(X_train, y_train, X_test, y_test):
    # Create and train the Support Vector Machines (SVM) model
    svc = SVC()
    svc.fit(X_train, y_train)

    # Make predictions on the test set
    y_pred_svc = svc.predict(X_test)

    # Calculate training accuracy
    svc_train = round(svc.score(X_train, y_train) * 100, 2)

    # Calculate accuracy score on the test set
    svc_accuracy = round(accuracy_score(y_pred_svc, y_test) * 100, 2)

    # Print results
    print("Training Accuracy    :", svc_train, "%")
    print("Model Accuracy Score :", svc_accuracy, "%")
    print("Classification_Report: \n", classification_report(y_test, y_pred


    plt.show()

# Example usage
support_vector_machine_model(X_train, y_train, X_test, y_test)
```

```
Training Accuracy    : 96.64 %
Model Accuracy Score : 96.05 %
Classification_Report:
               precision    recall  f1-score   support

           1       0.96      1.00      0.98      1921
           2       0.00      0.00      0.00        19
           3       0.00      0.00      0.00         9
           4       0.00      0.00      0.00        16
           5       0.00      0.00      0.00         3
           6       0.00      0.00      0.00        32

    accuracy                           0.96      2000
   macro avg       0.16      0.17      0.16      2000
weighted avg       0.92      0.96      0.94      2000
```

In [48]:
```
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_accura
    ]
})
```

In [50]:
```
```

In [ ]:
```
```