In [8]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

sns.set_style("darkgrid")
```

In [9]:
```
data= pd.read_csv("predictive_maintenance.csv")
```

In [10]:
```
data.head(50)
```

Out[10]:

	UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Target	Failure Type
0	1	M14860	M	298.1	308.6	1551	42.8	0	0	No Failure
1	2	L47181	L	298.2	308.7	1408	46.3	3	0	No Failure
2	3	L47182	L	298.1	308.5	1498	49.4	5	0	No Failure
3	4	L47183	L	298.2	308.6	1433	39.5	7	0	No Failure
4	5	L47184	L	298.2	308.7	1408	40.0	9	0	No Failure
5	6	M14865	M	298.1	308.6	1425	41.9	11	0	No Failure
6	7	L47186	L	298.1	308.6	1558	42.4	14	0	No Failure
7	8	L47187	L	298.1	308.6	1527	40.2	16	0	No Failure
8	9	M14868	M	298.3	308.7	1667	28.6	18	0	No Failure
9	10	M14869	M	298.5	309.0	1741	28.0	21	0	No Failure
10	11	H29424	H	298.4	308.9	1782	23.9	24	0	No Failure
11	12	H29425	H	298.6	309.1	1423	44.3	29	0	No Failure
12	13	M14872	M	298.6	309.1	1339	51.1	34	0	No Failure
13	14	M14873	M	298.6	309.2	1742	30.0	37	0	No Failure
14	15	L47194	L	298.6	309.2	2035	19.6	40	0	No Failure
15	16	L47195	L	298.6	309.2	1542	48.4	42	0	No Failure
16	17	M14876	M	298.6	309.2	1311	46.6	44	0	No Failure
17	18	M14877	M	298.7	309.2	1410	45.6	47	0	No Failure
18	19	H29432	H	298.8	309.2	1306	54.5	50	0	No Failure
19	20	M14879	M	298.9	309.3	1632	32.5	55	0	No Failure
20	21	H29434	H	298.9	309.3	1375	42.7	58	0	No Failure
21	22	L47201	L	298.8	309.3	1450	44.8	63	0	No Failure
22	23	M14882	M	298.9	309.3	1581	30.7	65	0	No Failure
23	24	L47203	L	299.0	309.4	1758	25.7	68	0	No Failure

	UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Target	Failure Type
24	25	M14884	M	299.0	309.4	1561	37.3	70	0	No Failure
25	26	L47205	L	299.0	309.5	1861	23.3	73	0	No Failure
26	27	L47206	L	299.1	309.5	1512	39.0	75	0	No Failure
27	28	H29441	H	299.1	309.4	1811	24.6	77	0	No Failure
28	29	L47208	L	299.1	309.4	1439	44.2	82	0	No Failure
29	30	L47209	L	299.0	309.4	1693	30.1	84	0	No Failure
30	31	M14890	M	299.1	309.5	1339	48.2	86	0	No Failure
31	32	L47211	L	299.0	309.4	1798	25.5	89	0	No Failure
32	33	L47212	L	299.0	309.4	1419	48.3	91	0	No Failure
33	34	L47213	L	298.9	309.3	1665	32.5	93	0	No Failure
34	35	M14894	M	298.8	309.1	1559	34.7	95	0	No Failure
35	36	M14895	M	298.8	309.2	1452	48.6	98	0	No Failure
36	37	M14896	M	298.9	309.2	1581	36.7	101	0	No Failure
37	38	L47217	L	298.8	309.1	1439	39.2	104	0	No Failure
38	39	H29452	H	298.9	309.2	1379	50.7	106	0	No Failure
39	40	L47219	L	298.8	309.1	1350	52.5	111	0	No Failure
40	41	L47220	L	298.8	309.1	1362	45.4	113	0	No Failure
41	42	L47221	L	298.8	309.1	1368	50.8	115	0	No Failure
42	43	M14902	M	298.8	309.1	1368	49.1	117	0	No Failure
43	44	H29457	H	298.8	309.2	1372	48.5	120	0	No Failure
44	45	M14904	M	298.8	309.1	1472	47.5	125	0	No Failure
45	46	L47225	L	298.8	309.1	1489	49.1	128	0	No Failure
46	47	M14906	M	298.7	309.0	1843	25.8	130	0	No Failure
47	48	L47227	L	298.8	309.1	1418	46.3	133	0	No Failure

	UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Target	Failure Type
48	49	H29462	H	298.8	309.2	1425	53.9	135	0	No Failure
49	50	M14909	M	298.9	309.2	1412	44.1	140	0	No Failure

In [11]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 10 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   UDI                      10000 non-null  int64  
 1   Product ID               10000 non-null  object 
 2   Type                     10000 non-null  object 
 3   Air temperature [K]      10000 non-null  float64
 4   Process temperature [K]  10000 non-null  float64
 5   Rotational speed [rpm]   10000 non-null  int64  
 6   Torque [Nm]              10000 non-null  float64
 7   Tool wear [min]          10000 non-null  int64  
 8   Target                   10000 non-null  int64  
 9   Failure Type             10000 non-null  object 
dtypes: float64(3), int64(4), object(3)
memory usage: 781.4+ KB
```

In [13]:
```
data= data.drop(["UDI",'Product ID'],axis=1)
data.head(3)
```

Out[13]:

	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

In [16]:
```
data.groupby(['Target','Failure Type']).count().drop(['Process temperature [K]','Rotational speed [rpm]','Torque [Nm]','Tool wear
```

Out[16]:

Target	Failure Type	count
0	No Failure	9643
	Random Failures	18
1	Heat Dissipation Failure	112
	No Failure	9
	Overstrain Failure	78
	Power Failure	95
	Tool Wear Failure	45

In [20]:

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

data.groupby(['Target','Failure Type']).median()
```

Out[20]:

Target	Failure Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]
0	No Failure	300.00	310.0	1507.0	39.80	107.0
	Random Failures	300.75	311.1	1490.0	44.60	142.0
1	Heat Dissipation Failure	302.45	310.7	1346.0	52.35	106.0
	No Failure	300.50	309.9	1438.0	45.20	119.0
	Overstrain Failure	299.45	310.1	1362.5	56.75	207.0
	Power Failure	300.40	310.2	1386.0	63.60	100.0
	Tool Wear Failure	300.40	310.3	1521.0	37.70	215.0

In [21]:

```
data.groupby(['Type','Target']).median()
```

Out[21]:

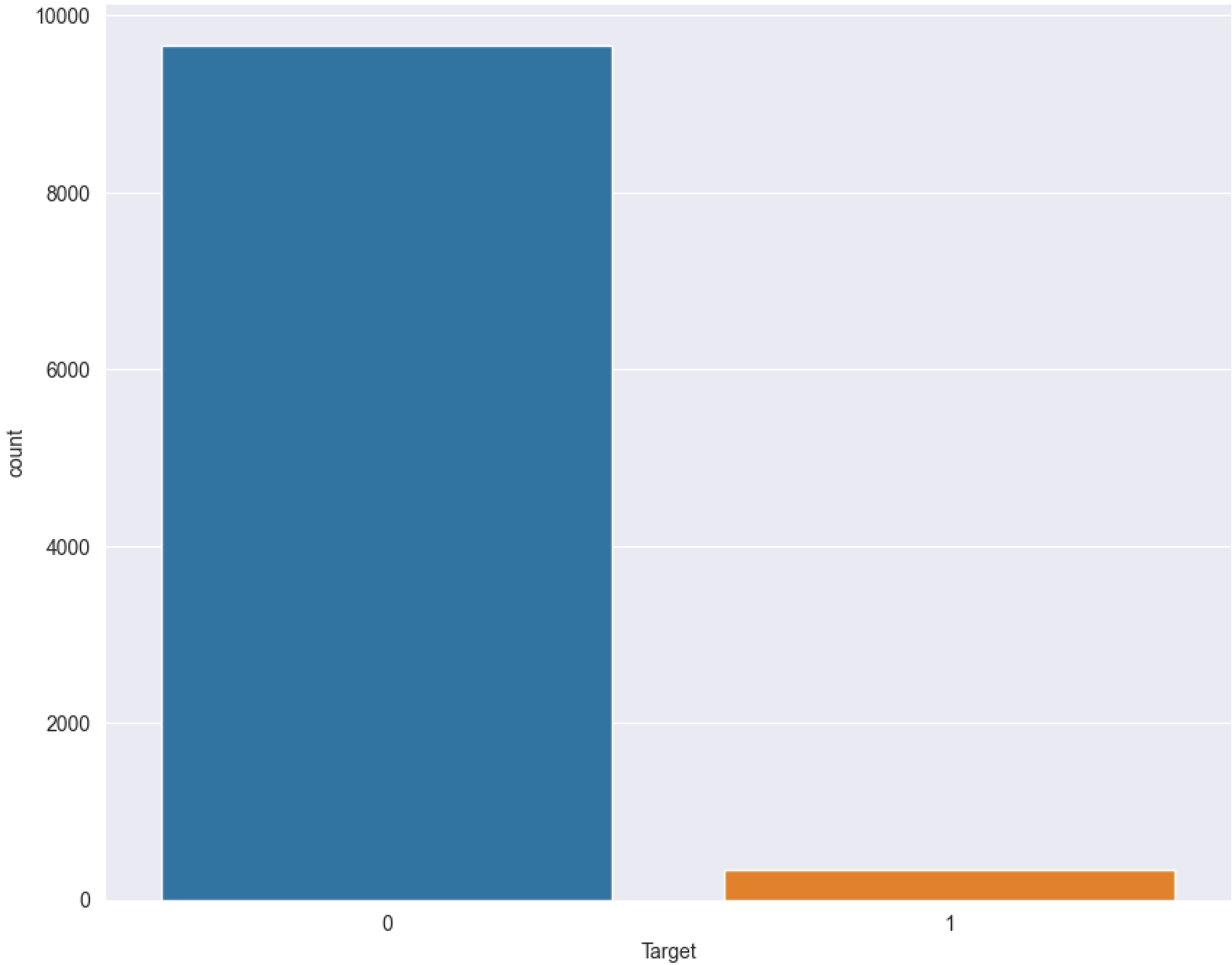
Type	Target	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]
H	0	299.7	309.9	1502.0	40.2	106.0
	1	302.0	310.2	1371.0	53.8	147.0
L	0	300.1	310.1	1508.0	39.7	107.0
	1	301.2	310.4	1362.0	53.9	182.0
M	0	300.1	310.0	1506.0	40.0	105.0
	1	302.0	310.6	1372.0	51.6	125.0

In [22]:

```
plt.figure(figsize=(10,8))
sns.countplot(data=data,x='Target')
```

Out[22]:

```
<AxesSubplot: xlabel='Target', ylabel='count'>
```



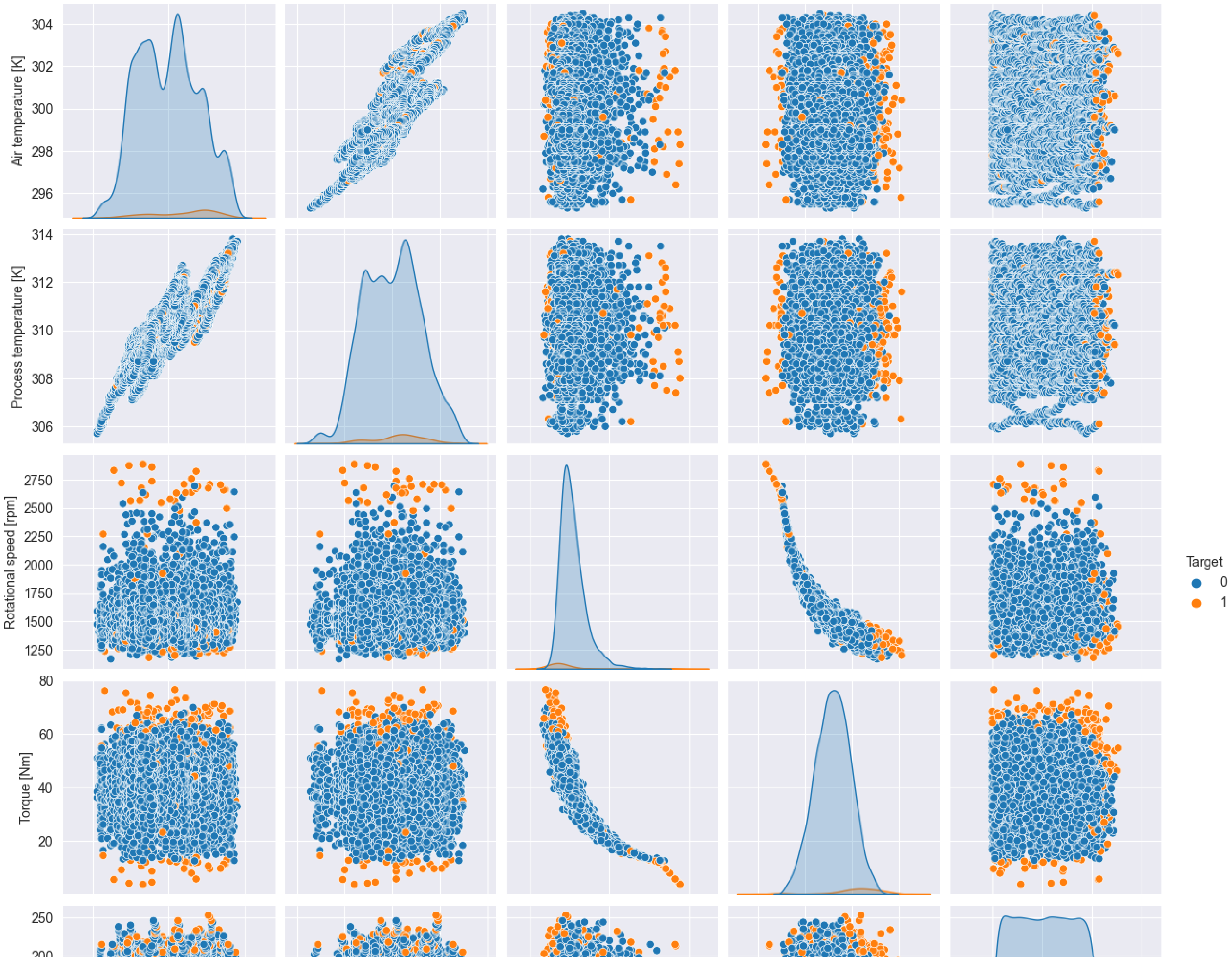
In [23]:
```
plt.figure(figsize=(10,5))
```

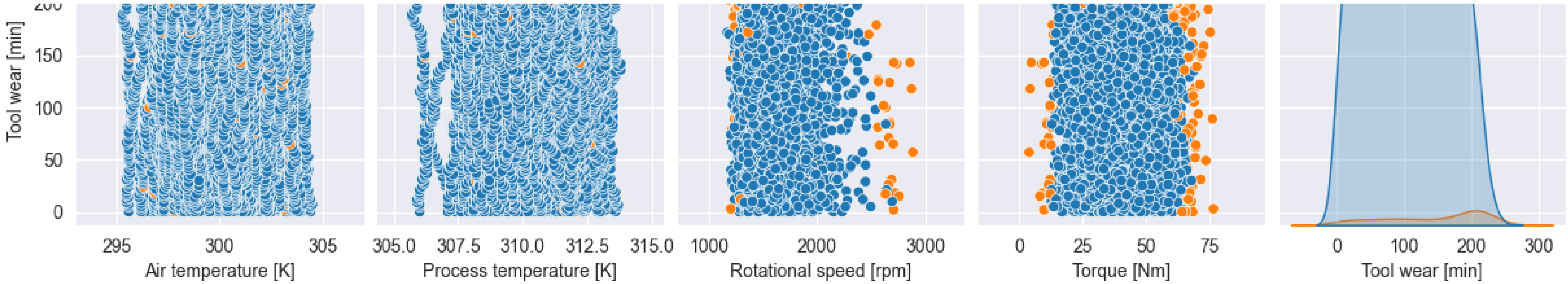
```
sns.countplot(data=data[data['Target']==1],x='Failure Type')
```

Out[23]: <AxesSubplot: xlabel='Failure Type', ylabel='count'>

In [24]:
```
sns.pairplot(data,hue='Target')
```

Out[24]: <seaborn.axisgrid.PairGrid at 0x178f038b6a0>

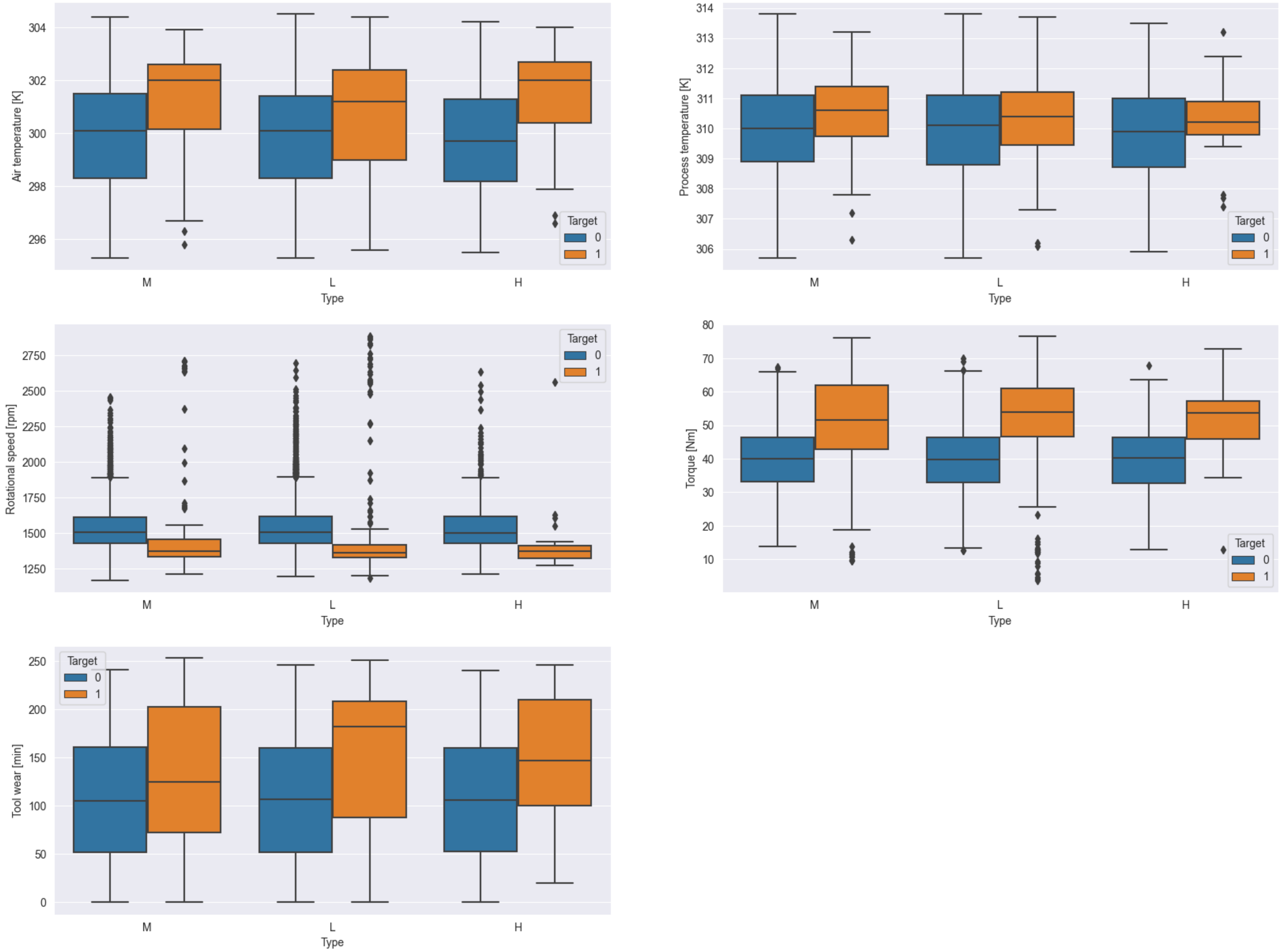

Air temperature [K]
Process temperature [K]
Rotational speed [rpm]
Torque [Nm]
Target
0
1



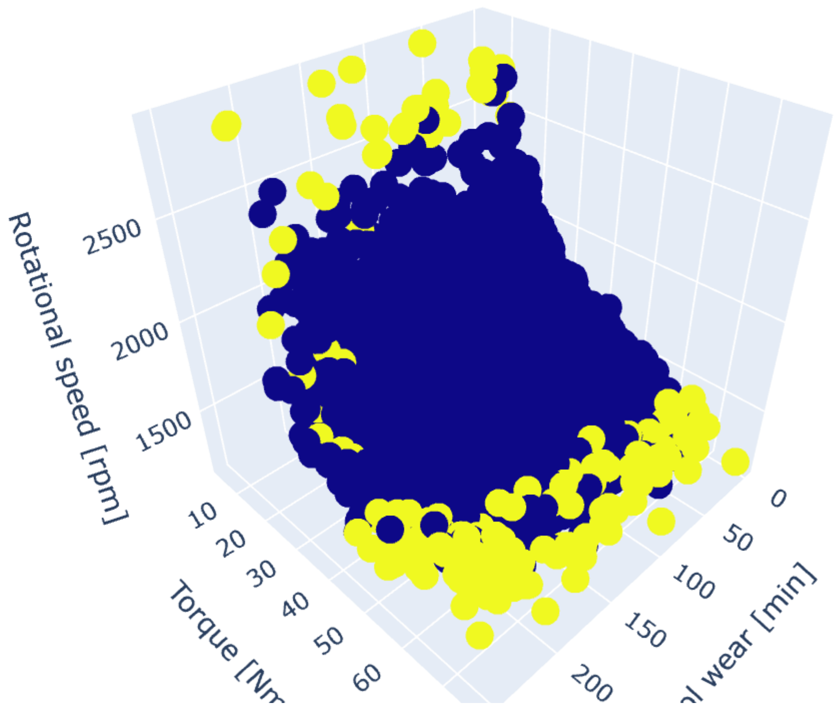
In [25]:
```
data.columns
```

Out[25]:
```
Index(['Type', 'Air temperature [K]', 'Process temperature [K]',
       'Rotational speed [rpm]', 'Torque [Nm]', 'Tool wear [min]', 'Target',
       'Failure Type'],
      dtype='object')
```

In [27]:
```
plt.figure(figsize=(20,15))
m=1
for i in ['Air temperature [K]', 'Process temperature [K]',
         'Rotational speed [rpm]', 'Torque [Nm]', 'Tool wear [min]']:
    plt.subplot(3,2,m)
    sns.boxplot(data=data,y=i,x='Type',hue='Target')
    m+=1
```


Air temperature [K]
Process temperature [K]
Rotational speed [rpm]
Torque [Nm]
Tool wear [min]
Type
M
L
H
Target
0
1

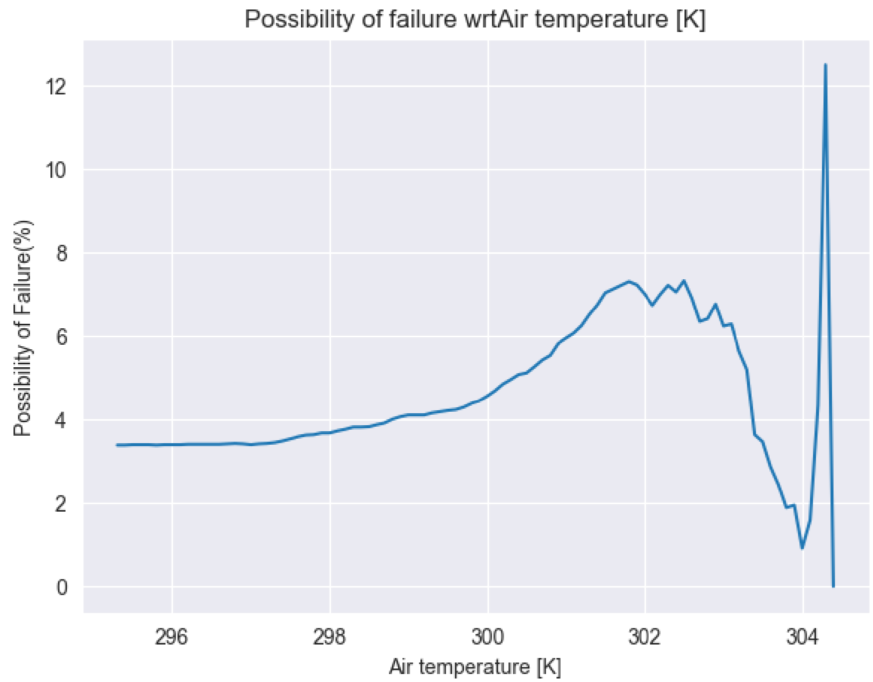
In [30]:
```
import plotly.express as px
fig=px.scatter_3d(data,x='Tool wear [min]',y='Torque [Nm]',z='Rotational speed [rpm]',color='Target')
fig.show()
```

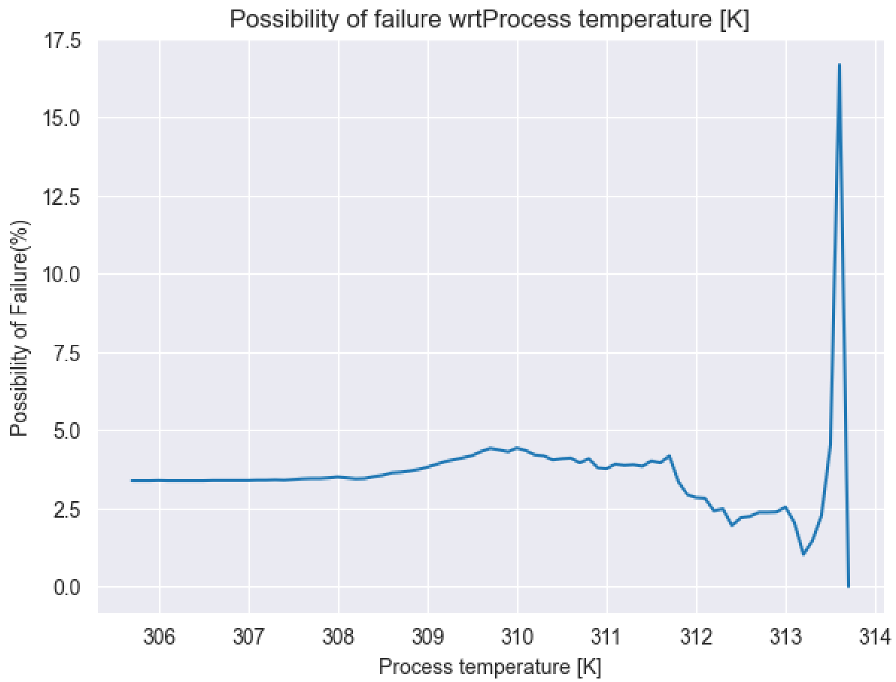


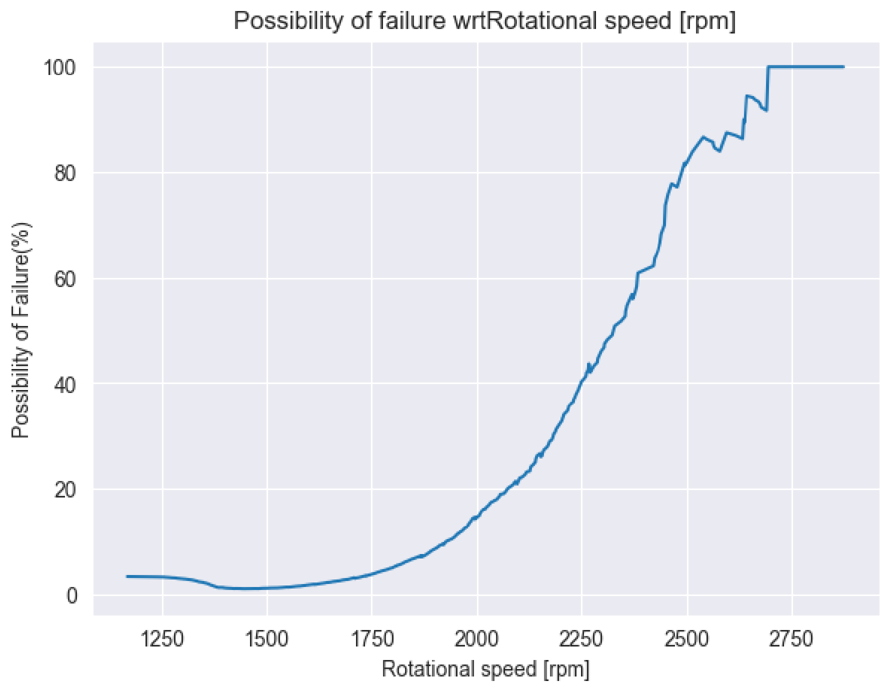
In [31]:
```
def feat_prob(feature,data):
    x,y=[],[]
    for j in data[feature].unique():
        temp=data
```

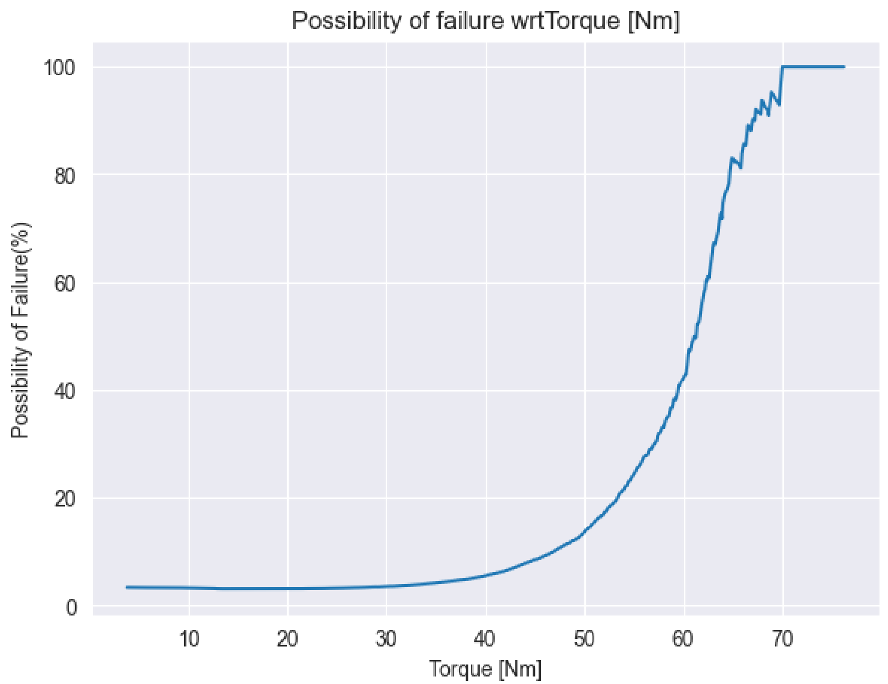
```
        temp = temp[temp[feature]>j]
        y.append(round((temp.Target.mean()*100),2))
        x.append(j)
    return(x,y)
```

In [33]:
```
plt.figure(figsize=(15,17))
m=1
for i in ['Air temperature [K]', 'Process temperature [K]',
       'Rotational speed [rpm]', 'Torque [Nm]', 'Tool wear [min]']:
    plt.subplot(3,2,m).set_title(label=("Possibility of failure wrt"+i))
    x,y = feat_prob(i,data)
    plt.xlabel(i)
    plt.ylabel("Possibility of Failure(%)")
    sns.lineplot(y=y,x=x)
    m+=1
```

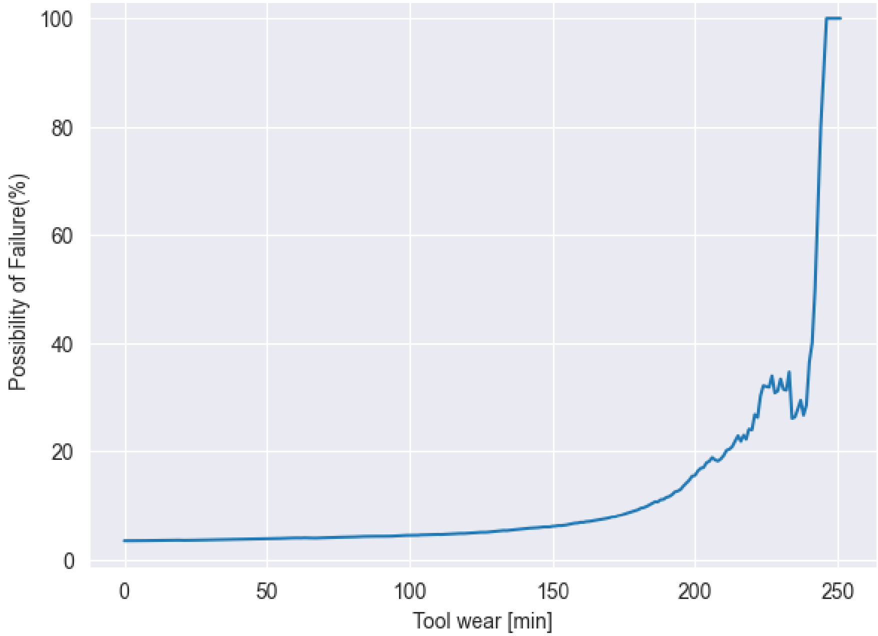






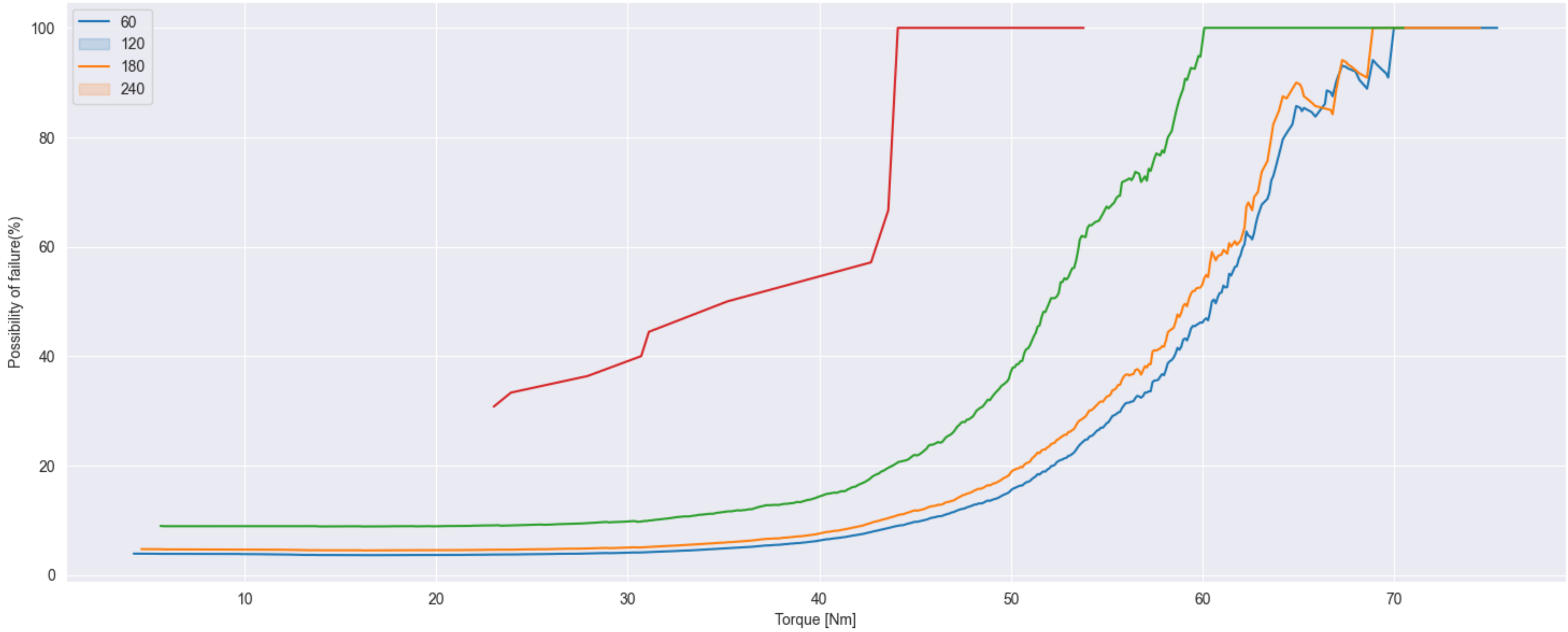


Possibility of failure wrtTool wear [min]

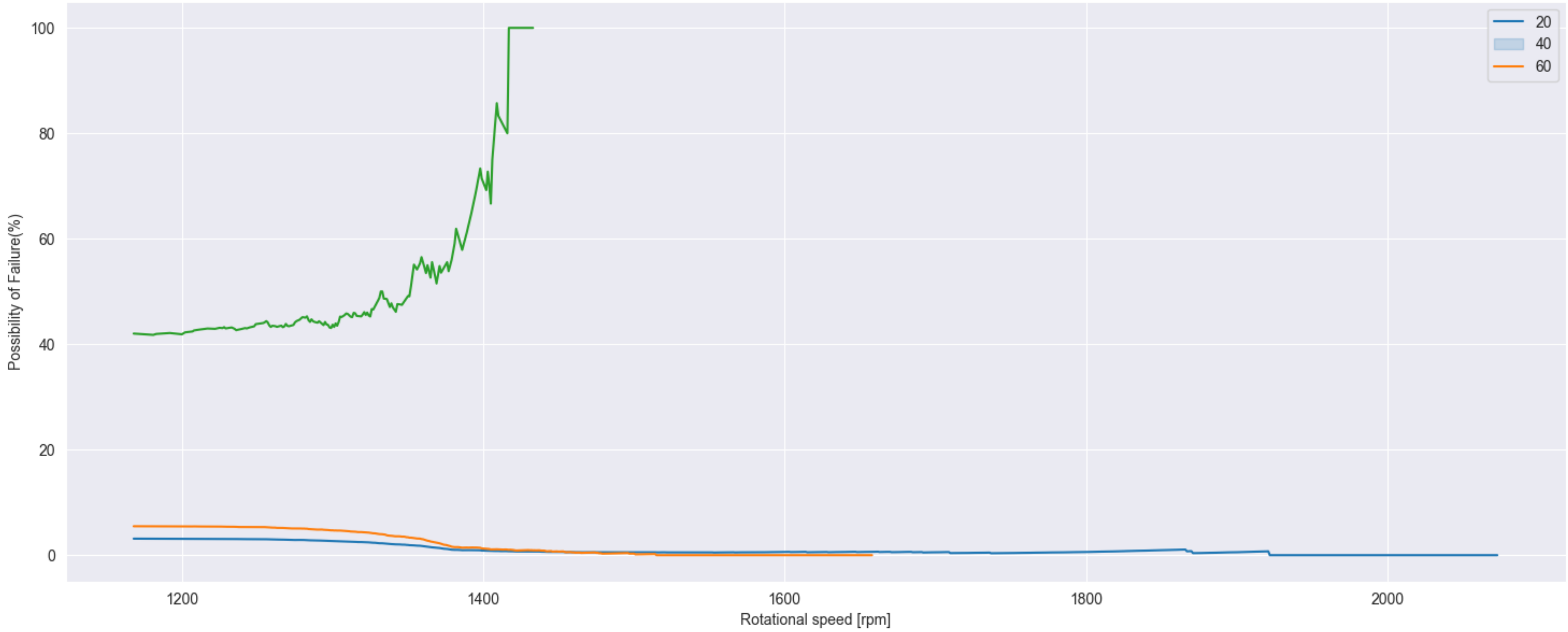


In [34]:
```
plt.figure(figsize=(18,7))
m=1

datasets = []
for i in [60,120,180,240]:
    datasets.append(data[data['Tool wear [min]']>=i])

for i in datasets:
    x,y = feat_prob('Torque [Nm]',i)
    plt.xlabel('Torque [Nm]')
    plt.ylabel("Possibility of failure(%)")
    sns.lineplot(y=y,x=x,legend='brief')

    m+=1

plt.legend([60,120,180,240])
```

Out[34]: <matplotlib.legend.Legend at 0x178fa23ae30>

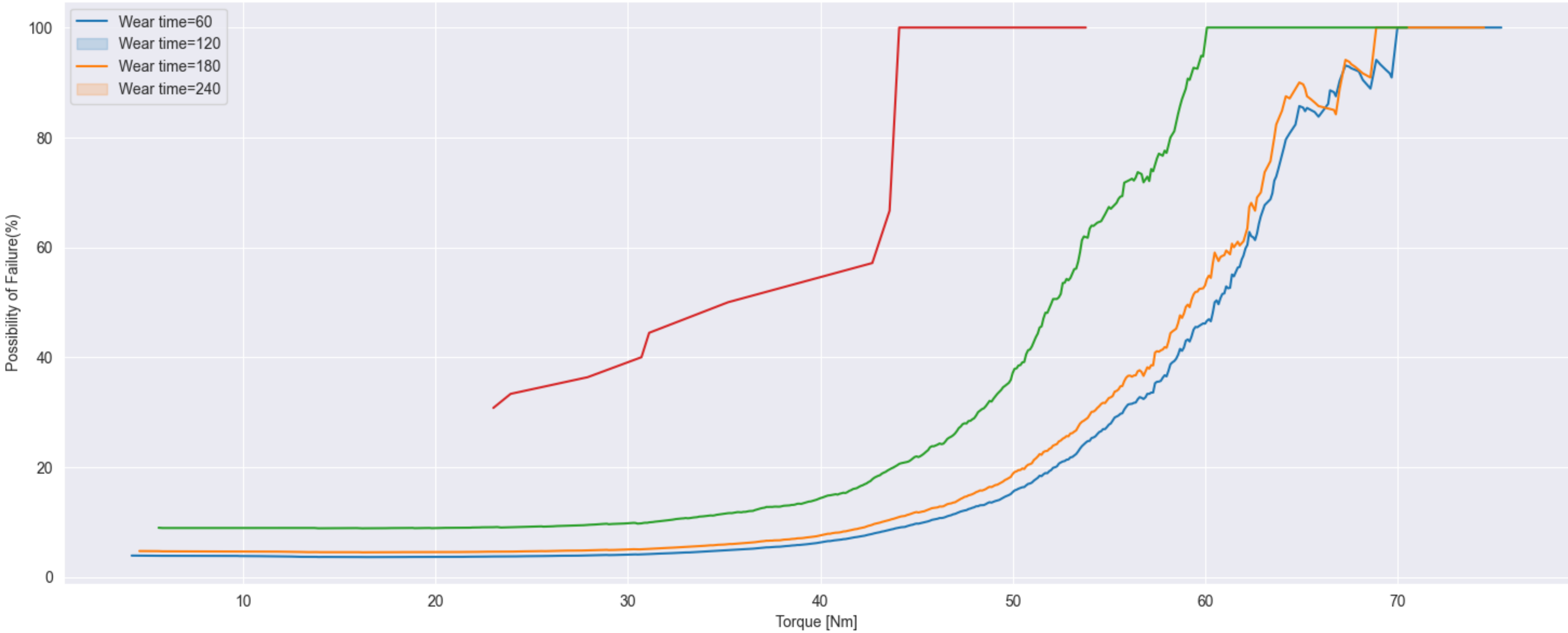


In [37]:
```
plt.figure(figsize=(18,7))
m=1

datasets= []
for i in [20,40,60]:
    datasets.append(data[data['Torque [Nm]']>=i])
for i in datasets:
    x,y = feat_prob('Rotational speed [rpm]',i)
    plt.xlabel('Rotational speed [rpm]')
    plt.ylabel('Possibility of Failure(%)')
    sns.lineplot(y=y,x=x,legend='brief')

    m+=1
    plt.legend([20,40,60])
```



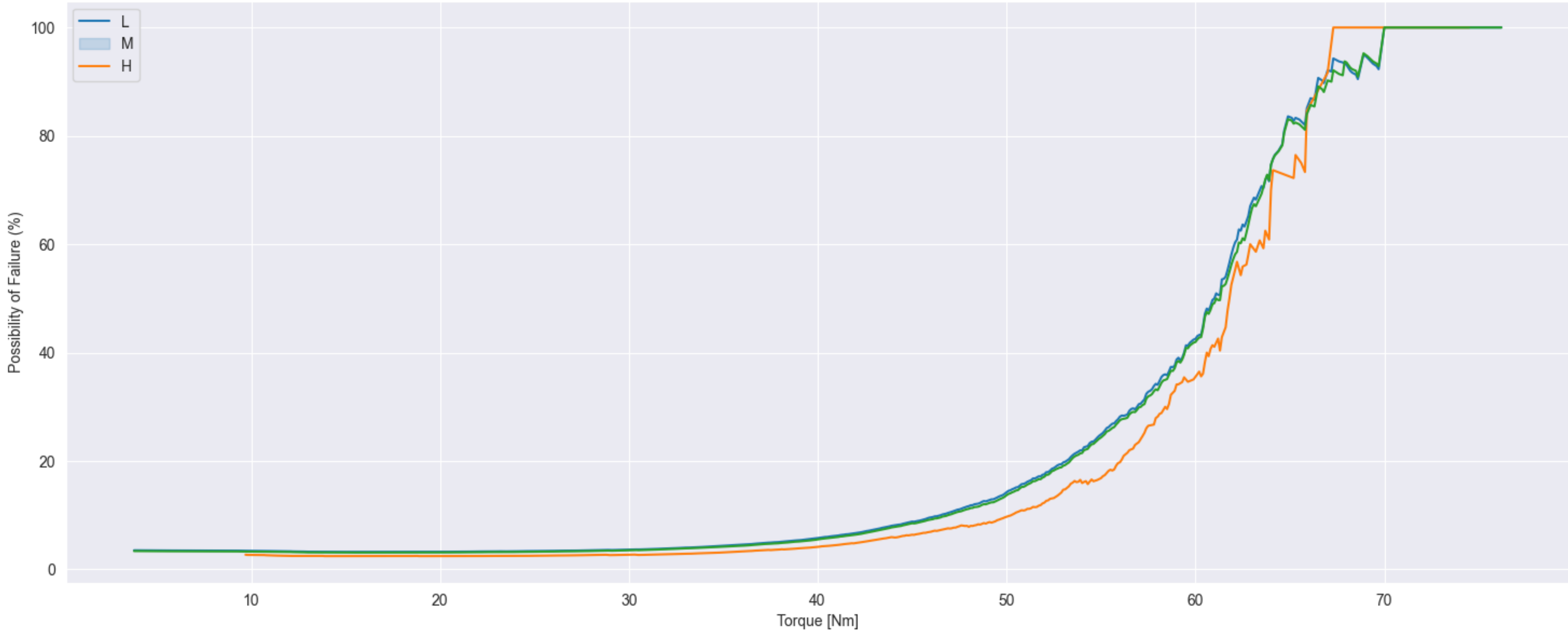
In [43]:
```
plt.figure(figsize=(18,7))
m=1
datasets =[]
for i in [60,120,180,240]:
        datasets.append(data[data['Tool wear [min]']>=i])

for i in datasets:
        x,y = feat_prob('Torque [Nm]',i)
        plt.xlabel('Torque [Nm]')
        plt.ylabel('Possibility of Failure(%)')
        sns.lineplot(y=y,x=x,legend='brief')

        m+=1

plt.legend(["Wear time="+"60",
                "Wear time="+"120",
                "Wear time="+"180",
                "Wear time="+"240"])
```

Out[43]: <matplotlib.legend.Legend at 0x178faedb2e0>



In [44]:
```
plt.figure(figsize=(18,7))
m=1

datasets = []
for i in ['L','M','H']:
    datasets.append(data[data['Type']>=i])

for i in datasets :
    x,y = feat_prob("Torque [Nm]",i)
    plt.xlabel("Torque [Nm]")
    plt.ylabel("Possibility of Failure (%)")
    sns.lineplot(y=y,x=x,legend='brief')

    m+=1

plt.legend(['L','M','H'])
```

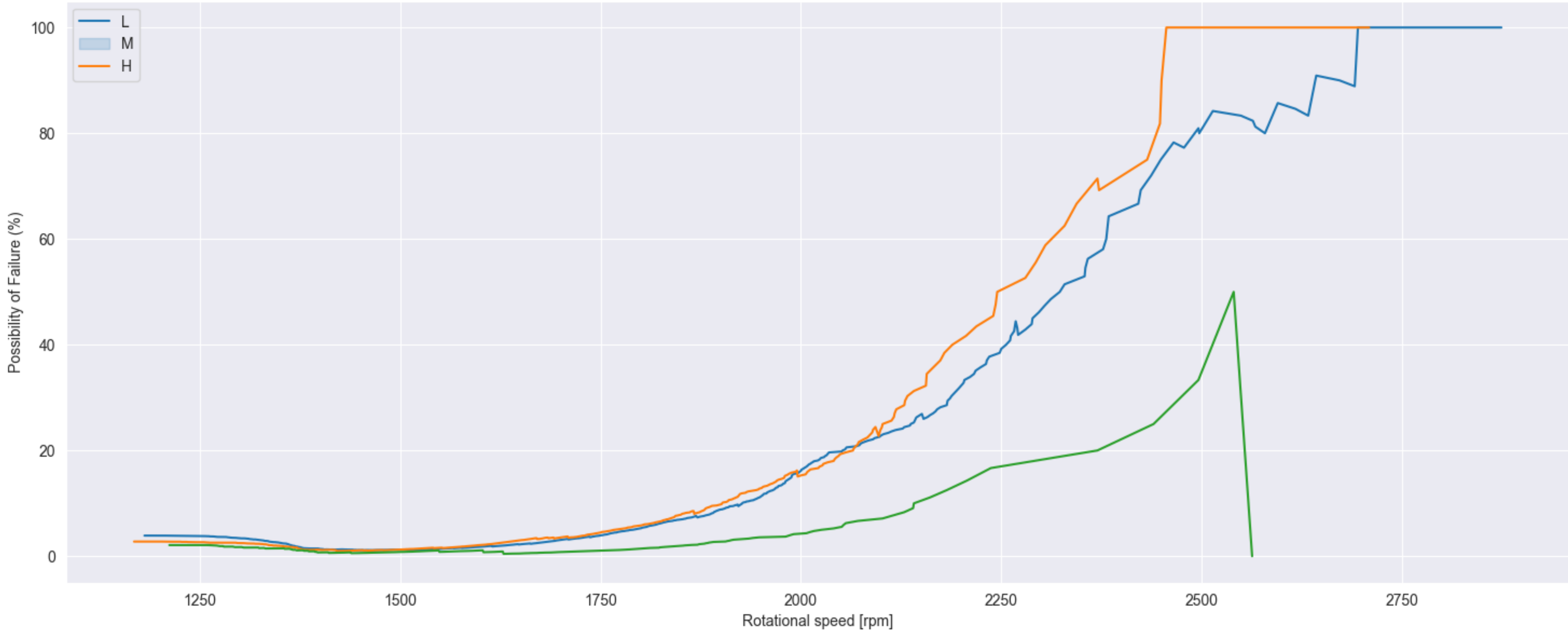
Out[44]: <matplotlib.legend.Legend at 0x178faaa2ce0>



In [47]:

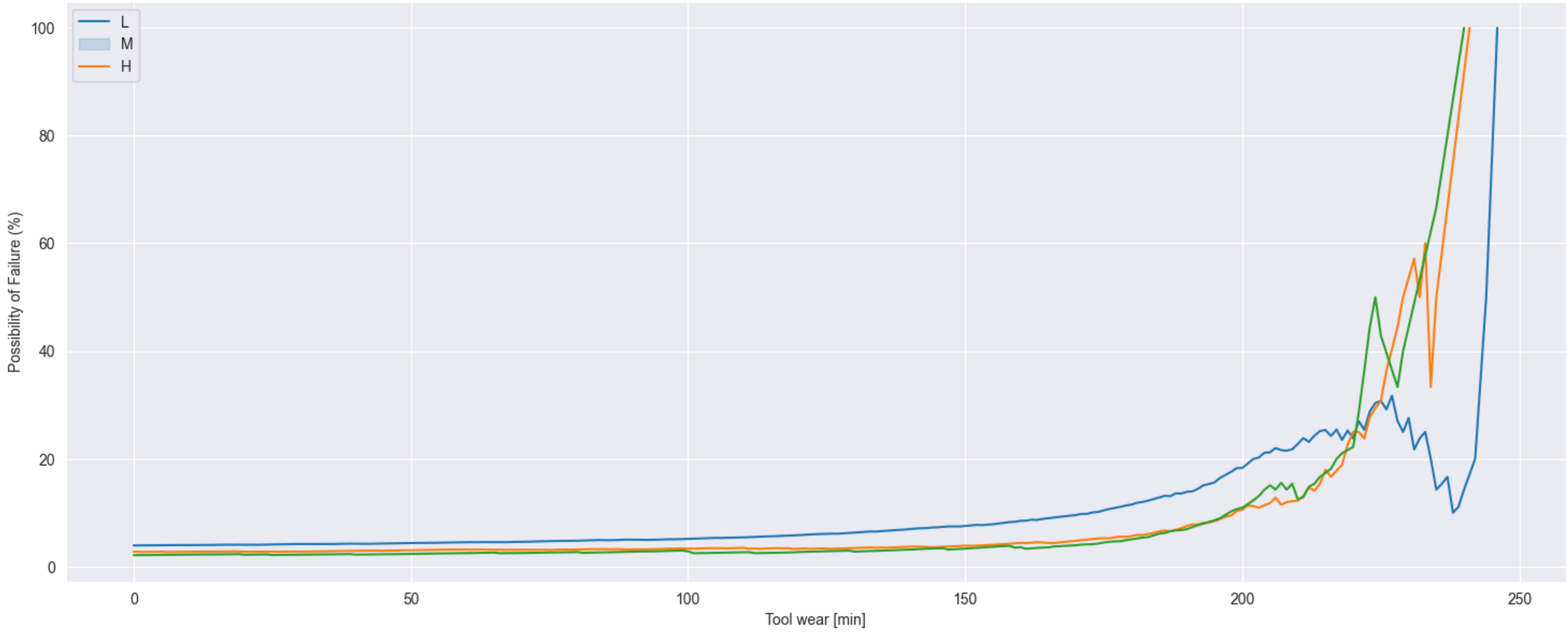
```
plt.figure(figsize=(18,7))
m=1

datasets = []
for i in ['L','M','H']:
    datasets.append(data[data['Type']==i])

for i in datasets:
    x,y = feat_prob('Rotational speed [rpm]',i)
    plt.xlabel('Rotational speed [rpm]')
    plt.ylabel('Possibility of Failure (%)')
    sns.lineplot(y=y,x=x,legend='brief')

    m+=1
plt.legend(['L','M','H'])
```

Out[47]: <matplotlib.legend.Legend at 0x178fb76d9f0>



In [48]:
```python
plt.figure(figsize=(18,7))
m=1

datasets=[]
for i in ['L','M','H']:
    datasets.append(data[data['Type']==i])
for i in datasets:
    x,y= feat_prob("Tool wear [min]",i)
    plt.xlabel("Tool wear [min]")
    plt.ylabel("Possibility of Failure (%)")
    sns.lineplot(y=y,x=x,legend='brief')

    m+=1
plt.legend(['L','M','H'])
```

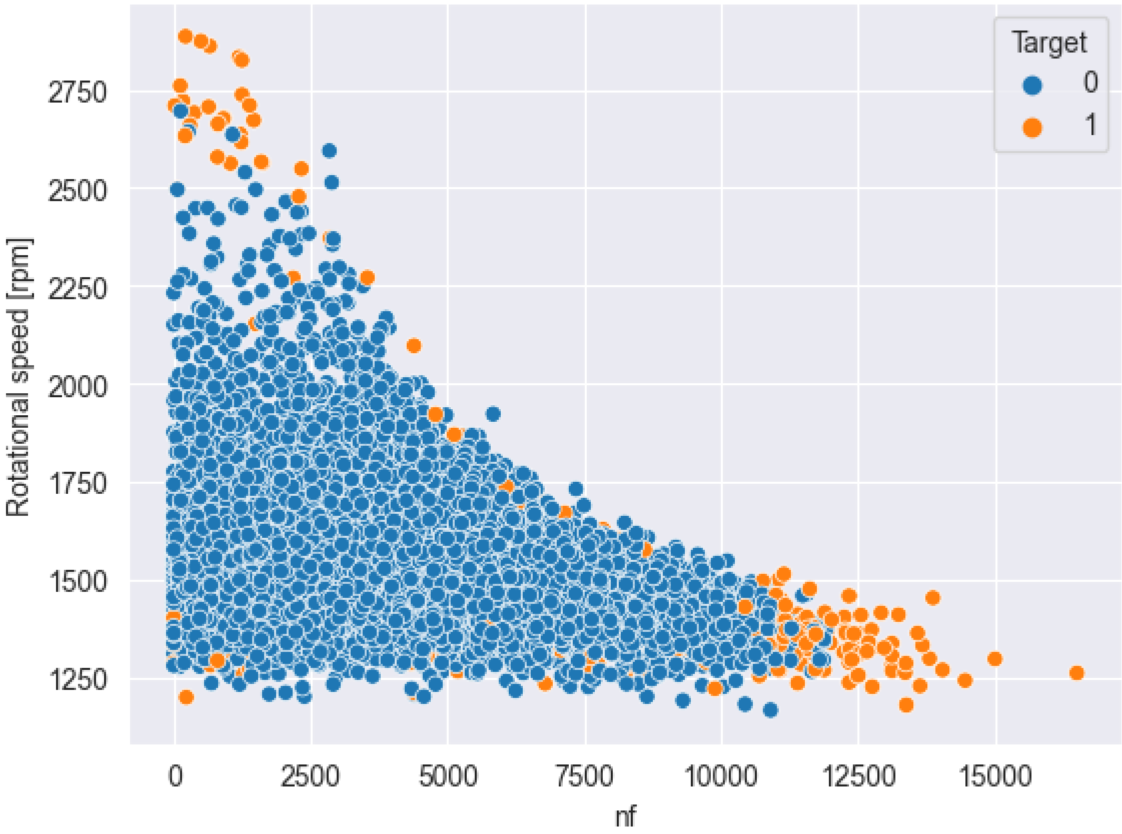
Out[48]: <matplotlib.legend.Legend at 0x178fc3712a0>



In [49]:
```
data['nf'] = data['Tool wear [min]']*data['Torque [Nm]']
```

In [50]:
```
sns.scatterplot(data=data,x='nf',y='Rotational speed [rpm]',hue= 'Target')
```

Out[50]: <AxesSubplot: xlabel='nf', ylabel='Rotational speed [rpm]'>



In [51]:
```
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()

label_encoder.fit(data['Type'])
data['Type'] = label_encoder.transform(data['Type'])

label_encoder.fit(data['Target'])
data['Target']=label_encoder.transform(data['Target'])
```

In [52]:
```
data.tail()
```

Out[52]:

	Type	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Target	Failure Type	nf
9995	2	298.8	308.4	1604	29.5	14	0	No Failure	413.0
9996	0	298.9	308.4	1632	31.8	17	0	No Failure	540.6
9997	2	299.0	308.6	1645	33.4	22	0	No Failure	734.8
9998	0	299.0	308.7	1408	48.5	25	0	No Failure	1212.5
9999	2	299.0	308.7	1500	40.2	30	0	No Failure	1206.0

In [56]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test=train_test_split(data.drop(['Failure Type','Target'],axis=1),data['Target'],test_size=0.3,randon
```

In [58]:

```
import time

from sklearn.metrics import accuracy_score,classification_report
classifier=[]
imported_as=[]

#LGBM
import lightgbm as lgb
lgbm = lgb.LGBMClassifier()
classifier.append('LightGBM')
imported_as.append('lgbm')

#MultiLayerPerceptron
from sklearn.neural_network import MLPClassifier
mlp=MLPClassifier()
classifier.append('Multi Layer Perceptron')
imported_as.append('mlp')

#Bagging
from sklearn.ensemble import BaggingClassifier
bc = BaggingClassifier()
classifier.append('Bagging')
imported_as.append('bc')

#GBC
from sklearn.ensemble import GradientBoostingClassifier
gbc = GradientBoostingClassifier()
```

```
classifier.append('Gradient Boosting')
imported_as.append('gbc')

#ADA
from sklearn.ensemble import AdaBoostClassifier
ada = AdaBoostClassifier()
classifier.append('Ada Boost')
imported_as.append('ada')

#XGB
import xgboost as xgb
from xgboost import XGBClassifier
xgb = XGBClassifier()
classifier.append('XG Boost')
imported_as.append('xgb')

# Logistic Regression
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
classifier.append('Logistic Regression')
imported_as.append('lr')

#RFC
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
classifier.append('Random Forest')
imported_as.append('rfc')

#KNN
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=1)
classifier.append('k Nearest Neighbours')
imported_as.append('knn')

#SVM
from sklearn.svm import SVC
svc = SVC()
classifier.append('Support Vector Machine')
imported_as.append('svc')

#Grid
from sklearn.model_selection import GridSearchCV
```

```
param_grid = {'C': [0.1,1, 10, 100, 1000,2000], 'gamma': [1,0.1,0.01,0.001,0.0001], 'kernel': ['rbf']}
grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=3)
classifier.append('SVM tuning grid')
imported_as.append('grid')

#STcaking
from sklearn.ensemble import StackingClassifier
estimators=[('rf', RandomForestClassifier(n_estimators=10, random_state=42)),
            ('svr',SVC(random_state=42))]
stc = StackingClassifier(estimators=estimators, final_estimator=LogisticRegression())
classifier.append('Stacked (RFR & SVM)')
imported_as.append('stc')

classifiers = pd.DataFrame({'Classifier':classifier,'Imported as':imported_as})
print('All Models Imported\nModels stored in dataframe called classifiers')
```

```
All Models Imported
Models stored in dataframe called classifiers
```

In [59]:

```
class Modelling:
    def __init__(self, X_train, Y_train, X_test, Y_test, models):
        self.X_train = X_train
        self.X_test = X_test
        self.Y_train = Y_train
        self.Y_test = Y_test
        self.models = models

    def fit(self):
        model_acc = []
        model_time= []
        for i in self.models:
            start=time.time()
            if i == 'knn':
                accuracy = []
                for j in range(1,200):
                    kn = KNeighborsClassifier(n_neighbors=j)
                    kn.fit(self.X_train,self.Y_train)
                    predK = kn.predict(self.X_test)
                    accuracy.append([accuracy_score(self.Y_test,predK),j])
                temp = accuracy[0]
                for m in accuracy:
                    if temp[0] < m[0]:
```

```
                temp=m
            i = KNeighborsClassifier(n_neighbors=temp[1])
        i.fit(self.X_train,self.Y_train)
        model_acc.append(accuracy_score(self.Y_test,i.predict(self.X_test)))
        stop=time.time()
        model_time.append((stop-start))
        print(i,'has been fit')
    self.models_output = pd.DataFrame({'Models':self.models,'Accuracy':model_acc,'Runtime (s)':model_time})

def results(self):
    models=self.models_output
    models = models.sort_values(by=['Accuracy','Runtime (s)'],ascending=[False,True]).reset_index().drop('index',axis=1)
    self.best = models['Models'][0]
    models['Models']=models['Models'].astype(str).str.split("(", n = 2, expand = True)[0]
    models['Accuracy']=models['Accuracy'].round(5)*100
    self.models_output_cleaned=models
    return(models)

def best_model(self,type):
    if type=='model':
        return(self.best)
    elif type=='name':
        return(self.models_output_cleaned['Models'][0])

def best_model_accuracy(self):
    return(self.models_output_cleaned['Accuracy'][0])

def best_model_runtime(self):
    return(round(self.models_output_cleaned['Runtime (s)'][0],3))

def best_model_predict(self,X_test):
    return(self.best.predict(X_test))

def best_model_clmatrix(self):
    return(classification_report(self.Y_test,self.best.predict(self.X_test)))
```

In [60]:
```
display(classifier)
```

```
['LightGBM',
 'Multi Layer Perceptron',
 'Bagging',
 'Gradient Boosting',
 'Ada Boost',
 'XG Boost',
 'Logistic Regression',
 'Random Forest',
 'k Nearest Neighbours',
 'Support Vector Machine',
 'SVM tuning grid',
 'Stacked (RFR & SVM)']
```

In [61]:

```
models_to_test = [bc,gbc,ada,rfc,mlp,lr,knn,stc]
```

In [62]:

```
X_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7000 entries, 9069 to 7270
Data columns (total 7 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Type                     7000 non-null   int32  
 1   Air temperature [K]      7000 non-null   float64
 2   Process temperature [K]  7000 non-null   float64
 3   Rotational speed [rpm]   7000 non-null   int64  
 4   Torque [Nm]              7000 non-null   float64
 5   Tool wear [min]          7000 non-null   int64  
 6   nf                       7000 non-null   float64
dtypes: float64(4), int32(1), int64(2)
memory usage: 410.2 KB
```

In [63]:

```
classification = Modelling(X_train,y_train,X_test,y_test,models_to_test)
classification.fit()
```

```
BaggingClassifier() has been fit
GradientBoostingClassifier() has been fit
AdaBoostClassifier() has been fit
RandomForestClassifier() has been fit
MLPClassifier() has been fit
LogisticRegression() has been fit
KNeighborsClassifier(n_neighbors=1) has been fit
StackingClassifier(estimators=[('rf',
                                RandomForestClassifier(n_estimators=10,
                                                       random_state=42)),
                               ('svr', SVC(random_state=42))],
                   final_estimator=LogisticRegression()) has been fit
```

In [64]:
```
classification.results()
```

Out[64]:

	Models	Accuracy	Runtime (s)
0	BaggingClassifier	99.033	0.309766
1	RandomForestClassifier	98.867	0.940216
2	StackingClassifier	98.867	1.630276
3	GradientBoostingClassifier	98.800	1.192892
4	AdaBoostClassifier	97.667	0.387894
5	LogisticRegression	97.333	0.071944
6	KNeighborsClassifier	96.067	0.070685
7	MLPClassifier	94.133	1.297064

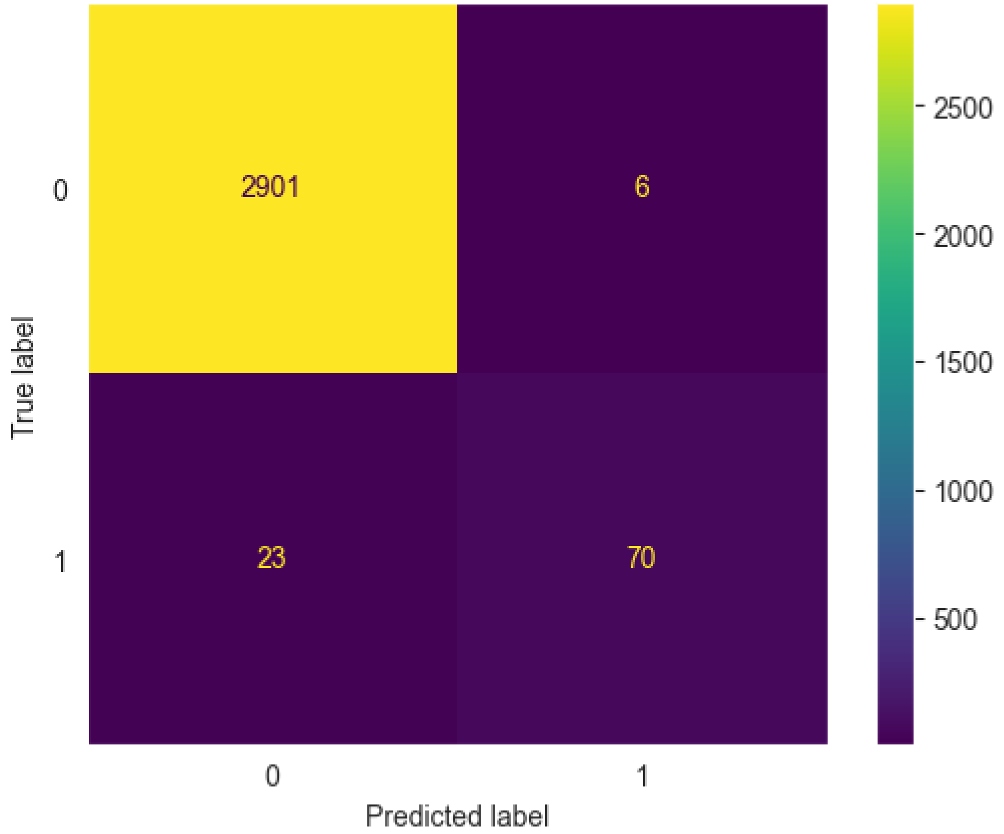
In [65]:
```
print('BestModel is:',  classification.best_model(type='name'))
print('Accuracy of model:',classification.best_model_accuracy())
print('Training Runtime in seconds',classification.best_model_runtime())
print('Classification Matrix:\n')
print(classification.best_model_clmatrix())
```

```
BestModel is: BaggingClassifier
Accuracy of model: 99.033
Training Runtime in seconds 0.31
Classification Matrix:

              precision    recall  f1-score   support

           0       0.99      1.00      1.00      2907
           1       0.92      0.75      0.83        93

    accuracy                           0.99      3000
   macro avg       0.96      0.88      0.91      3000
weighted avg       0.99      0.99      0.99      3000
```

In [66]:

```
sns.set_style("darkgrid", {"grid.color": "1", "grid.linestyle": " "})

from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(classification.best_model(type='model'), X_test, y_test)
```

Out[66]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x178f03356c0>


2901
6
23
70
0
1
True label
0
1
Predicted label
2500
2000
1500
1000
500

In [ ]: