Arian mohammad khani- 810603136

homework 3

The goal of this exercise is to evaluate the performance of binary and multiclass classification methods. To achieve this, the case of health monitoring and fault detection of a milling machine tool has been considered. The dataset in question (`machine_milling.csv` file) contains 1000 records, where each record indicates the condition of the milling tool (sixth column) based on five features: **"ambient temperature," "process temperature," "rotational speed of the tool," "torque applied to the tool axis,"** and **"duration of tool exposure to wear"** (first to fifth columns).

In [1]:
```
from pandas import *
from numpy import *
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import  train_test_split, GridSearchCV
from sklearn.preprocessing import  LabelEncoder, StandardScaler, MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
from imblearn.over_sampling import SMOTE
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.multiclass import OneVsRestClassifier, OneVsOneClassifier
```

a

In [2]:
```
data_frame = read_csv('milling_machine.csv')
data_frame.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 6 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Air Temp (°C)           9965 non-null   float64
 1   Process Temp (°C)       9990 non-null   float64
 2   Rotational Speed (RPM)  10000 non-null  float64
 3   Torque (Nm)             10000 non-null  float64
 4   Tool Wear (Seconds)     9993 non-null   float64
 5   Failure Types           9991 non-null   object
dtypes: float64(5), object(1)
memory usage: 468.9+ KB
```

In [3]:
```
data_frame.head()
```

Out[3]:

	Air Temp (°C)	Process Temp (°C)	Rotational Speed (RPM)	Torque (Nm)	Tool Wear (Seconds)	Failure Types
0	29.021640	71.620737	1515.840689	50.223021	664.638000	No Failure
1	21.886075	69.896471	2083.417786	52.221351	6628.080758	No Failure
2	29.020744	74.731134	2455.801496	57.822145	3295.576818	No Failure
3	25.793868	70.715109	2112.654324	69.910072	7116.479752	No Failure
4	21.056760	71.025092	1642.485295	68.411333	1191.996403	No Failure

In [4]:
```
data_frame.describe()
```

Out[4]:

	Air Temp (°C)	Process Temp (°C)	Rotational Speed (RPM)	Torque (Nm)	Tool Wear (Seconds)
count	9965.000000	9990.000000	10000.000000	10000.000000	9993.000000
mean	28.516926	80.812186	1401.909988	46.998845	11393.143344
std	7.719340	15.548350	968.446183	26.747646	9023.336380
min	20.001366	60.001876	0.047731	0.015920	3.469877
25%	23.176455	68.090324	423.672240	18.091381	5023.027818
50%	26.212082	76.553203	1377.047835	54.983239	8995.172952
75%	29.377536	92.825894	2307.969925	67.258375	15024.825673
max	49.998008	119.971025	2999.953724	89.993221	35999.566519

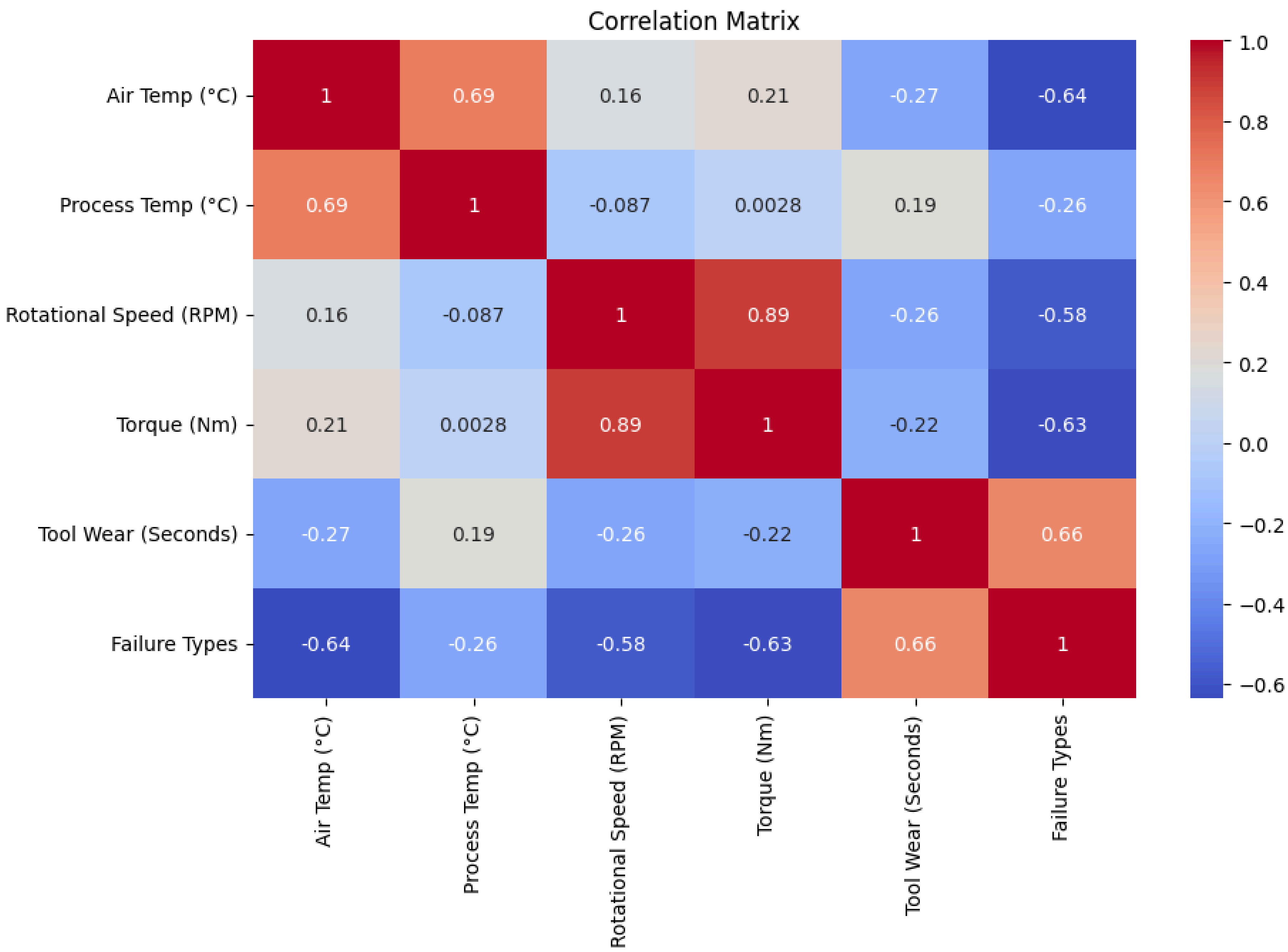
In [5]:
```
missing_v = data_frame.isnull().sum()
missing_v_ratio = data_frame.isnull().mean()

missing_data = DataFrame({
    'Missing value Count': missing_v,
    'Missing value Ratio': missing_v_ratio
})
missing_data
```

Out[5]:

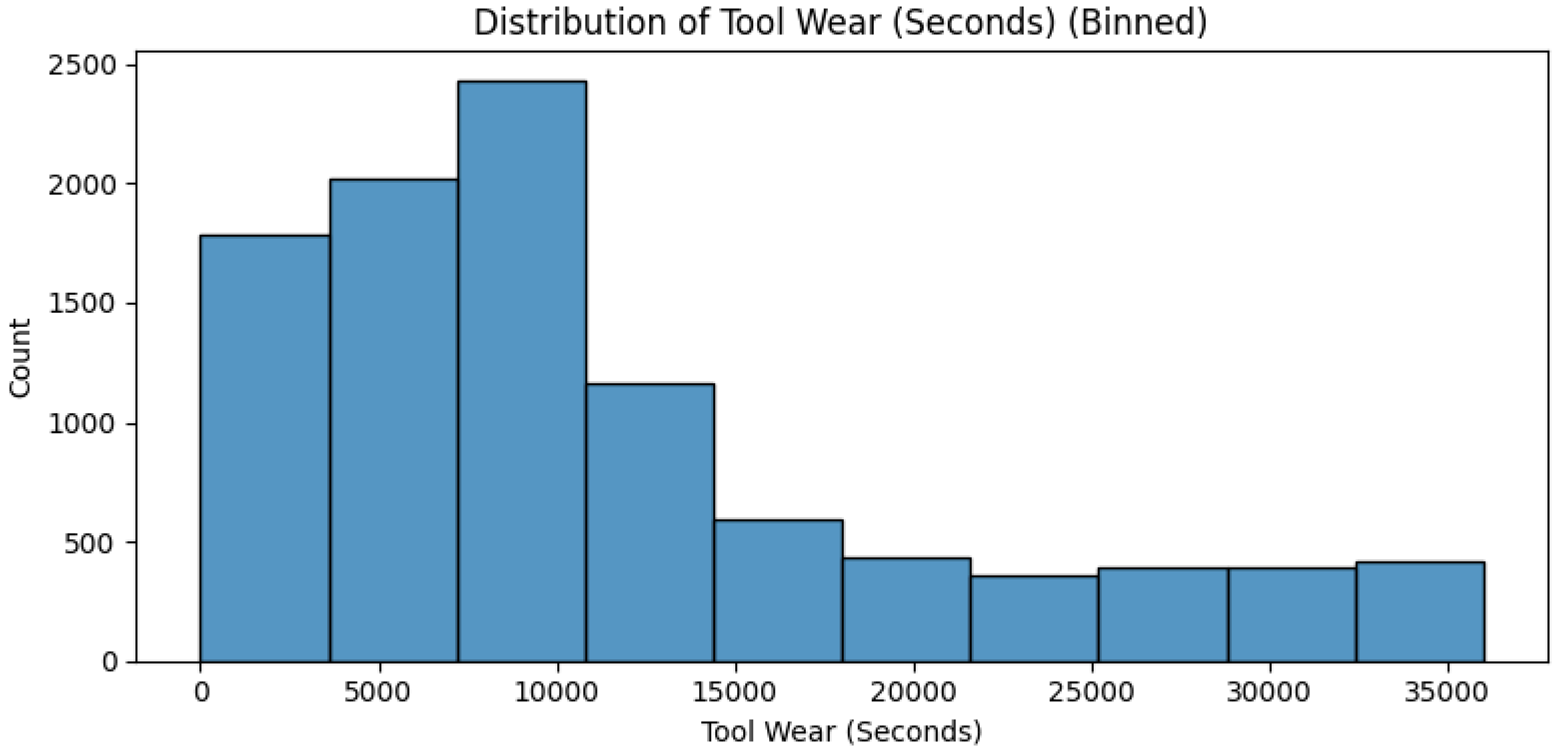
	Missing value Count	Missing value Ratio
Air Temp (°C)	35	0.0035
Process Temp (°C)	10	0.0010
Rotational Speed (RPM)	0	0.0000
Torque (Nm)	0	0.0000
Tool Wear (Seconds)	7	0.0007
Failure Types	9	0.0009

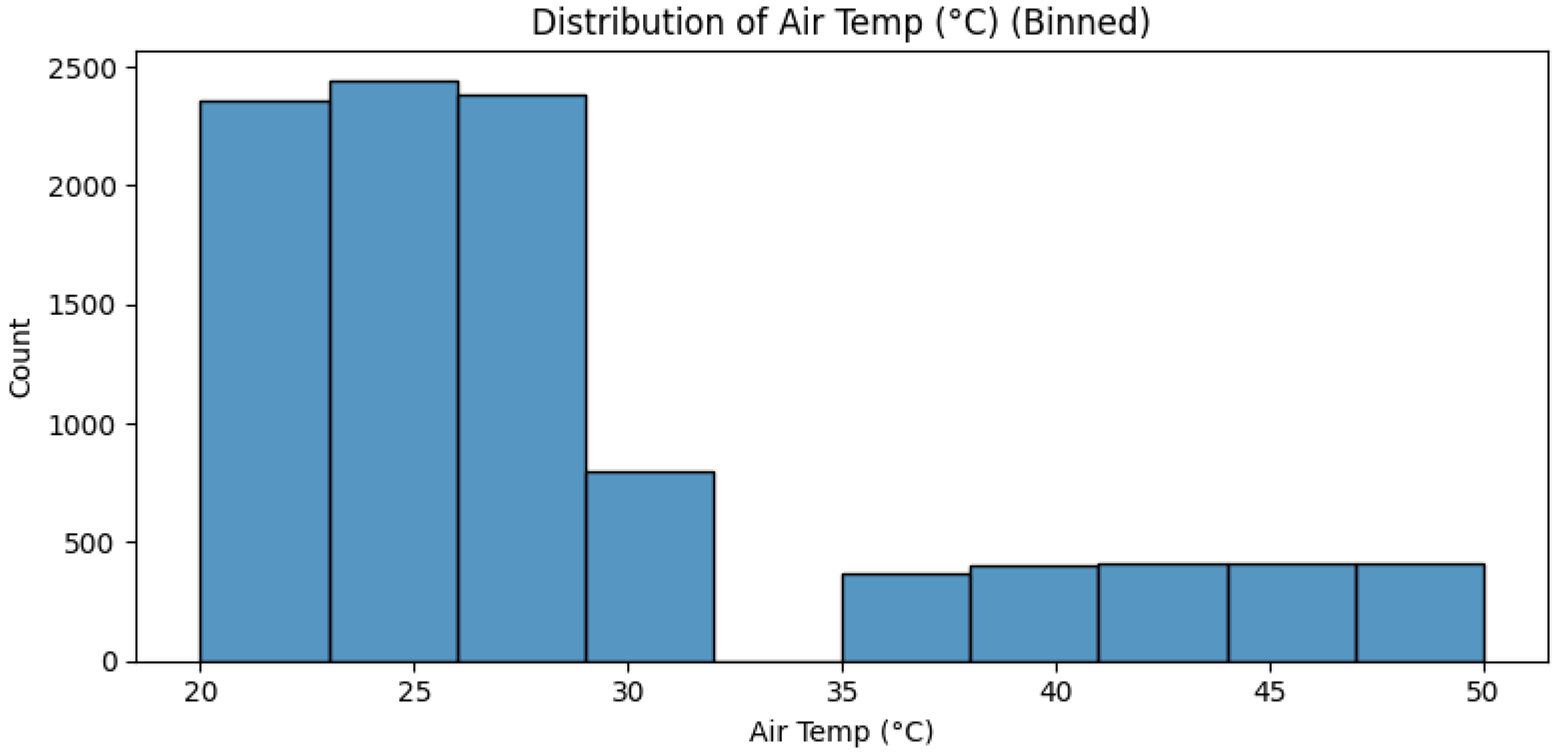
In [6]:
```
new_data = data_frame.copy()
label_encoder = LabelEncoder()
for column in new_data.select_dtypes(include='object').columns:
    new_data[column] = label_encoder.fit_transform(new_data[column])

correlation_matrix = new_data.corr(numeric_only=True)
correlation_with_failure = correlation_matrix['Failure Types']
sorted_corr = correlation_with_failure.sort_values(ascending=False)
print(sorted_corr)

plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```

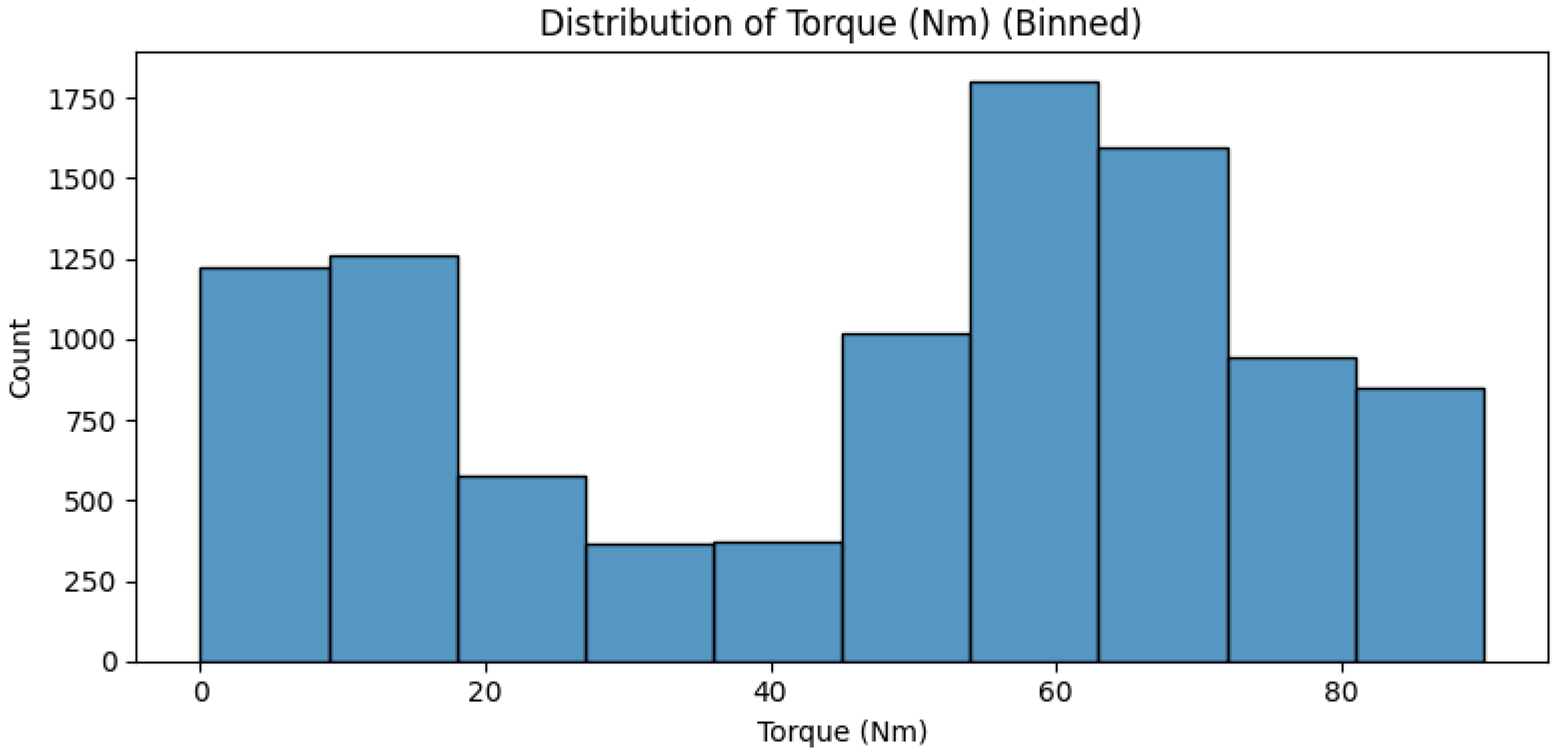
```
Failure Types             1.000000
Tool Wear (Seconds)       0.656459
Process Temp (°C)        -0.257862
Rotational Speed (RPM)   -0.582298
Torque (Nm)              -0.626631
Air Temp (°C)            -0.636946
Name: Failure Types, dtype: float64
```



In [7]:
```
top_features = sorted_corr.drop('Failure Types').abs().nlargest(3).index

for feature in top_features:
    plt.figure(figsize=(8, 4))
    sns.histplot(data=new_data, x=feature, bins=10, kde=False)
    plt.title(f'Distribution of {feature} (Binned)')
    plt.xlabel(feature)
    plt.ylabel('Count')
    plt.tight_layout()
    plt.show()
```







In [8]:
```
data_frame.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 6 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Air Temp (°C)           9965 non-null   float64
 1   Process Temp (°C)       9990 non-null   float64
 2   Rotational Speed (RPM)  10000 non-null  float64
 3   Torque (Nm)             10000 non-null  float64
 4   Tool Wear (Seconds)     9993 non-null   float64
 5   Failure Types           9991 non-null   object 
dtypes: float64(5), object(1)
memory usage: 468.9+ KB
```

b

In [9]:
```
for column in data_frame.columns:
    missing_value = data_frame[column].isnull().sum()
    if missing_value > 0:
        if data_frame[column].dtype == 'float64' :
            if  column != 'Failure Types':
                data_frame[column] = data_frame.groupby('Failure Types')[column].transform(lambda x: x.fillna(x.mean()))
        elif data_frame[column].dtype == 'object' :
            data_frame.dropna(subset=[column], inplace=True)
```

In [10]:
```
label_encoder = LabelEncoder()
for column in data_frame.select_dtypes(include='object').columns:
    data_frame[column] = label_encoder.fit_transform(data_frame[column])
```

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

```
<class 'pandas.core.frame.DataFrame'>
Index: 9991 entries, 0 to 9999
Data columns (total 6 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   Air Temp (°C)           9991 non-null   float64
 1   Process Temp (°C)       9991 non-null   float64
 2   Rotational Speed (RPM)  9991 non-null   float64
 3   Torque (Nm)             9991 non-null   float64
 4   Tool Wear (Seconds)     9991 non-null   float64
 5   Failure Types           9991 non-null   int32  
dtypes: float64(5), int32(1)
memory usage: 507.4 KB
```

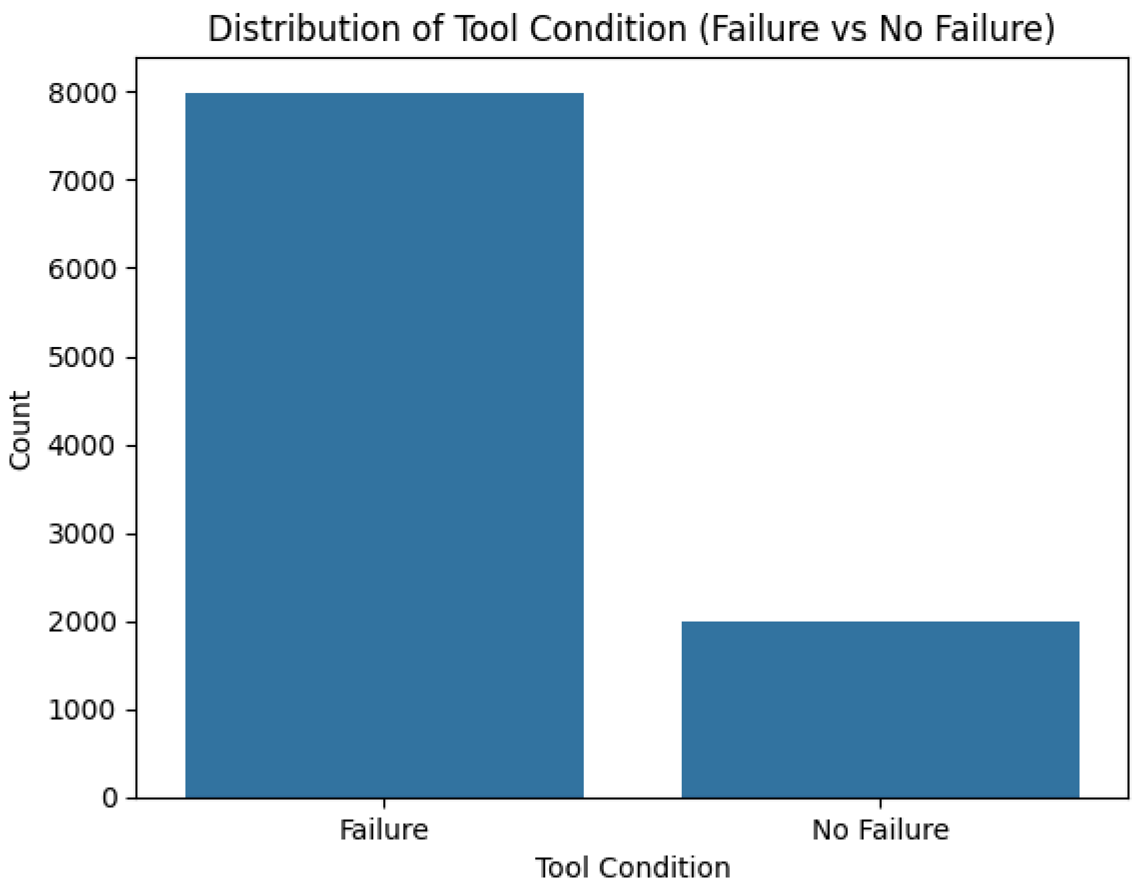
In [12]:
```
numeric_cols = ['Air Temp (°C)', 'Process Temp (°C)', 'Rotational Speed (RPM)', 'Torque (Nm)', 'Tool Wear (Seconds)']

scaler = StandardScaler()
data_frame[numeric_cols] = scaler.fit_transform(data_frame[numeric_cols])
```

c

In [13]:
```
data_frame['Failure_Binary'] = data_frame['Failure Types'].apply(lambda x: 'Failure' if x != 0 else 'No Failure')
data_frame['Failure_Binary'].value_counts()
```

Out[13]:
```
Failure_Binary
Failure       7993
No Failure    1998
Name: count, dtype: int64
```

In [14]:
```
sns.countplot(data=data_frame, x='Failure_Binary')
plt.title('Distribution of Tool Condition (Failure vs No Failure)')
plt.xlabel('Tool Condition')
plt.ylabel('Count')
plt.show()
```



In [15]:
```
X = data_frame.drop(columns=['Failure Types', 'Failure_Binary'])
y = data_frame['Failure_Binary']

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y_encoded)

print("After SMOTE:")
uniques, counts = unique(y_resampled, return_counts=True)
print(dict(zip(label_encoder.inverse_transform(uniques), counts)))


X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.2, random_state=42)
```

```
After SMOTE:
{'Failure': 7993, 'No Failure': 7993}
```

In [16]:
```
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "KNN": KNeighborsClassifier(n_neighbors=5),
    "SVM Linear": SVC(kernel='linear'),
    "SVM RBF": SVC(kernel='rbf')
}

results = []

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    acc = accuracy_score(y_test, y_pred)
    report = classification_report(y_test, y_pred, output_dict=True)
    conf_matrix = confusion_matrix(y_test, y_pred)

    print(f"\n{name}:\n")
    print("Confusion Matrix:\n", conf_matrix)
    print("Accuracy:", acc)
    print("Classification Report:\n", classification_report(y_test, y_pred))

    results.append({
        "Model": name,
        "Accuracy": acc,
        "Precision (Failure)": report['1']['precision'],
        "Recall (Failure)": report['1']['recall'],
        "F1-Score (Failure)": report['1']['f1-score']
    })
```

```
Logistic Regression:

Confusion Matrix:
 [[1572    0]
 [   0 1626]]
Accuracy: 1.0
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      1572
           1       1.00      1.00      1.00      1626

    accuracy                           1.00      3198
   macro avg       1.00      1.00      1.00      3198
weighted avg       1.00      1.00      1.00      3198


KNN:

Confusion Matrix:
 [[1572    0]
 [   0 1626]]
Accuracy: 1.0
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      1572
           1       1.00      1.00      1.00      1626

    accuracy                           1.00      3198
   macro avg       1.00      1.00      1.00      3198
weighted avg       1.00      1.00      1.00      3198


SVM Linear:

Confusion Matrix:
 [[1572    0]
 [   0 1626]]
Accuracy: 1.0
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      1572
           1       1.00      1.00      1.00      1626

    accuracy                           1.00      3198
   macro avg       1.00      1.00      1.00      3198
weighted avg       1.00      1.00      1.00      3198


SVM RBF:

Confusion Matrix:
 [[1572    0]
 [   0 1626]]
Accuracy: 1.0
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      1572
           1       1.00      1.00      1.00      1626

    accuracy                           1.00      3198
   macro avg       1.00      1.00      1.00      3198
weighted avg       1.00      1.00      1.00      3198
```

In [17]:

```python
# Logistic
param_grid_lr = {
    'C': [0.01, 0.1, 1, 10],
    'penalty': ['l2'],
}

grid_lr = GridSearchCV(LogisticRegression(max_iter=1000), param_grid_lr, cv=5, scoring='accuracy')
grid_lr.fit(X_train, y_train)

print("Best Params (LR):", grid_lr.best_params_)
```

```
Best Params (LR): {'C': 0.01, 'penalty': 'l2'}
```

In [18]:

```python
#KNN
param_grid_knn = {'n_neighbors': range(1, 21)}
grid_knn = GridSearchCV(KNeighborsClassifier(), param_grid_knn, cv=5)
grid_knn.fit(X_train, y_train)
print("Best K for KNN:", grid_knn.best_params_)
```

```
Best K for KNN: {'n_neighbors': 1}
```

In [19]:
```
#SVM
param_grid_svm_linear = {
    'C': [0.01, 0.1, 1, 10, 100],
    'kernel': ['linear']
}

grid_svm_linear = GridSearchCV(SVC(), param_grid_svm_linear, cv=5, scoring='accuracy')
grid_svm_linear.fit(X_train, y_train)

print("Best parameters for SVM (Linear):", grid_svm_linear.best_params_)
```

```
Best parameters for SVM (Linear): {'C': 0.01, 'kernel': 'linear'}
```

In [20]:
```
#SVM RBF
param_grid_svm = {
    'C': [0.1, 1, 10],
    'gamma': [1, 0.1, 0.01],
    'kernel': ['rbf']
}
grid_svm = GridSearchCV(SVC(), param_grid_svm, cv=5)
grid_svm.fit(X_train, y_train)
print("Best parameters for SVM (RBF):", grid_svm.best_params_)
```

```
Best parameters for SVM (RBF): {'C': 0.1, 'gamma': 1, 'kernel': 'rbf'}
```

In [21]:
```
comparison_table = DataFrame(results)
print(comparison_table.sort_values(by="Accuracy", ascending=False))
```

```
                 Model  Accuracy  Precision (Failure)  Recall (Failure)  \
0  Logistic Regression       1.0                  1.0               1.0
1                  KNN       1.0                  1.0               1.0
2           SVM Linear       1.0                  1.0               1.0
3              SVM RBF       1.0                  1.0               1.0

   F1-Score (Failure)
0                 1.0
1                 1.0
2                 1.0
3                 1.0
```

d

In [22]:
```
X_multi = data_frame.drop(columns=['Failure Types', 'Failure_Binary'])
y_multi = data_frame['Failure Types']

smote_multi = SMOTE(random_state=42)
X_res_multi, y_res_multi = smote_multi.fit_resample(X_multi, y_multi)


X_train_m, X_test_m, y_train_m, y_test_m = train_test_split(X_res_multi, y_res_multi, test_size=0.2, random_state=42)


models_multiclass = {
    "KNN": KNeighborsClassifier(n_neighbors=5),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(random_state=42),
    "SVM (One-vs-Rest)": OneVsRestClassifier(SVC(kernel='rbf')),
    "SVM (One-vs-one)":OneVsOneClassifier(SVC(kernel='linear', C=1))

}


results_multiclass = []

for name, model in models_multiclass.items():
    model.fit(X_train_m, y_train_m)
    y_pred_m = model.predict(X_test_m)

    acc = accuracy_score(y_test_m, y_pred_m)
    conf_matrix = confusion_matrix(y_test_m, y_pred_m)
    report = classification_report(y_test_m, y_pred_m, output_dict=True)

    print(f"\n{name}:\n")
    print("Confusion Matrix:\n", conf_matrix)
    print("Accuracy:", acc)
    print("Classification Report:\n", classification_report(y_test_m, y_pred_m))

    results_multiclass.append({
        "Model": name,
        "Accuracy": acc,
        "Macro Precision": report['macro avg']['precision'],
        "Macro Recall": report['macro avg']['recall'],
        "Macro F1": report['macro avg']['f1-score']
    })
```

```
KNN:

Confusion Matrix:
 [[418   0   0   0   0]
 [  0 410   2   0   0]
 [  0   0 392   0   0]
 [  0   0   0 387   0]
 [  0   0   0   0 390]]
Accuracy: 0.9989994997498749
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       418
           1       1.00      1.00      1.00       412
           2       0.99      1.00      1.00       392
           3       1.00      1.00      1.00       387
           4       1.00      1.00      1.00       390

    accuracy                           1.00      1999
   macro avg       1.00      1.00      1.00      1999
weighted avg       1.00      1.00      1.00      1999


Decision Tree:

Confusion Matrix:
 [[418   0   0   0   0]
 [  0 412   0   0   0]
 [  0   0 392   0   0]
 [  0   0   0 387   0]
 [  0   0   0   0 390]]
Accuracy: 1.0
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       418
           1       1.00      1.00      1.00       412
           2       1.00      1.00      1.00       392
           3       1.00      1.00      1.00       387
           4       1.00      1.00      1.00       390

    accuracy                           1.00      1999
   macro avg       1.00      1.00      1.00      1999
weighted avg       1.00      1.00      1.00      1999


Random Forest:

Confusion Matrix:
 [[418   0   0   0   0]
 [  0 412   0   0   0]
 [  0   0 392   0   0]
 [  0   0   0 387   0]
 [  0   0   0   0 390]]
Accuracy: 1.0
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       418
           1       1.00      1.00      1.00       412
           2       1.00      1.00      1.00       392
           3       1.00      1.00      1.00       387
           4       1.00      1.00      1.00       390

    accuracy                           1.00      1999
   macro avg       1.00      1.00      1.00      1999
weighted avg       1.00      1.00      1.00      1999


SVM (One-vs-Rest):

Confusion Matrix:
 [[418   0   0   0   0]
 [  0 412   0   0   0]
 [  0   0 392   0   0]
 [  0   0   0 387   0]
 [  0   0   0   0 390]]
Accuracy: 1.0
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       418
           1       1.00      1.00      1.00       412
           2       1.00      1.00      1.00       392
           3       1.00      1.00      1.00       387
           4       1.00      1.00      1.00       390
```

```
    accuracy                           1.00      1999
   macro avg       1.00      1.00      1.00      1999
weighted avg       1.00      1.00      1.00      1999


SVM (One-vs-one):

Confusion Matrix:
 [[418   0   0   0   0]
 [  0 412   0   0   0]
 [  0   0 392   0   0]
 [  0   0   0 387   0]
 [  0   0   0   0 390]]
Accuracy: 1.0
Classification Report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       418
           1       1.00      1.00      1.00       412
           2       1.00      1.00      1.00       392
           3       1.00      1.00      1.00       387
           4       1.00      1.00      1.00       390

    accuracy                           1.00      1999
   macro avg       1.00      1.00      1.00      1999
weighted avg       1.00      1.00      1.00      1999
```

In [23]:
```
#KNN
param_grid_knn = {'n_neighbors': range(1, 21)}
grid_knn_multi = GridSearchCV(KNeighborsClassifier(), param_grid_knn, cv=5)
grid_knn_multi.fit(X_train_m, y_train_m)
print("Best K for KNN (Multiclass):", grid_knn_multi.best_params_)
```

```
Best K for KNN (Multiclass): {'n_neighbors': 4}
```

In [24]:
```
# Decision Tree
param_grid_dt = {'max_depth': [3, 5, 10, None], 'min_samples_split': [2, 5, 10]}
grid_dt = GridSearchCV(DecisionTreeClassifier(random_state=42), param_grid_dt, cv=5)
grid_dt.fit(X_train_m, y_train_m)
print("Best params for Decision Tree:", grid_dt.best_params_)
```

```
Best params for Decision Tree: {'max_depth': 5, 'min_samples_split': 2}
```

In [25]:
```
# Random Forest
param_grid_rf = {'n_estimators': [50, 100, 150], 'max_depth': [None, 10, 20]}
grid_rf = GridSearchCV(RandomForestClassifier(random_state=42), param_grid_rf, cv=5)
grid_rf.fit(X_train_m, y_train_m)
print("Best params for Random Forest:", grid_rf.best_params_)
```

```
Best params for Random Forest: {'max_depth': None, 'n_estimators': 50}
```

In [26]:
```
# SVM One-vs-Rest
param_grid_svm_multi = {
    'estimator__C': [0.1, 1, 10],
    'estimator__gamma': [1, 0.1, 0.01]
}
grid_svm_multi = GridSearchCV(OneVsRestClassifier(SVC(kernel='rbf')), param_grid_svm_multi, cv=5)
grid_svm_multi.fit(X_train_m, y_train_m)
print("Best params for SVM (One-vs-Rest):", grid_svm_multi.best_params_)
```

```
Best params for SVM (One-vs-Rest): {'estimator__C': 10, 'estimator__gamma': 0.1}
```

In [27]:
```
# One-vs-One SVM
param_grid_svm_ovo = {
    'estimator__C': [0.1, 1, 10],
    'estimator__gamma': [1, 0.1, 0.01]
}
grid_svm_ovo = GridSearchCV(OneVsOneClassifier(SVC(kernel='rbf')),param_grid_svm_ovo,cv=5,scoring='accuracy')
grid_svm_ovo.fit(X_train_m, y_train_m)
print("Best params for SVM (One-vs-One):", grid_svm_ovo.best_params_)
```

```
Best params for SVM (One-vs-One): {'estimator__C': 0.1, 'estimator__gamma': 1}
```

In [29]:
```
comparison_multiclass = DataFrame(results_multiclass)
print(comparison_multiclass.sort_values(by="Accuracy", ascending=False))
```

```
               Model  Accuracy  Macro Precision  Macro Recall  Macro F1
1      Decision Tree  1.000000         1.000000      1.000000  1.000000
2      Random Forest  1.000000         1.000000      1.000000  1.000000
3  SVM (One-vs-Rest)  1.000000         1.000000      1.000000  1.000000
4   SVM (One-vs-one)  1.000000         1.000000      1.000000  1.000000
0                KNN  0.998999         0.998985      0.999029  0.999004
```