

```
# Install required packages
!pip install pandas scikit-learn xgboost matplotlib seaborn pyod

# Import all libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, accuracy_s
from xgboost import XGBClassifier
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from pyod.models.knn import KNN
import joblib
import random

# Set random seed for reproducibility
np.random.seed(42)
random.seed(42)
```

```

⇒ Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/di
Requirement already satisfied: xgboost in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist
Requirement already satisfied: seaborn in /usr/local/lib/python3.11/dist-pa
Collecting pyod
  Downloading pyod-2.0.5-py3-none-any.whl.metadata (46 kB)
    46.3/46.3 kB 1.7 MB/s eta 0:0
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/d
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/pyt
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/di
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/di
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/d
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/pytho
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.1
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.1
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/di
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.1
Requirement already satisfied: numba>=0.51 in /usr/local/lib/python3.11/dis
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-p
Downloading pyod-2.0.5-py3-none-any.whl (200 kB)
    200.6/200.6 kB 2.0 MB/s eta 0:0
Installing collected packages: pyod
Successfully installed pyod-2.0.5

```

```

# =====
# 1. Data Loading and Initial Exploration
# =====
print("STEP 1: DATA LOADING AND EXPLORATION")
print("=====")

# Load the dataset
#from google.colab import files
#uploaded = files.upload()

# Read the CSV file
df = pd.read_csv('/content/ai4i2020.csv')
print(f"\nDataset shape: {df.shape}")
print("\nFirst 5 rows:")
print(df.head())

# Basic info
print("\nDataset info:")

```

```
print(df.info())
```

➞ STEP 1: DATA LOADING AND EXPLORATION

Dataset shape: (10000, 14)

First 5 rows:

	UDI	Product ID	Type	Air temperature [K]	Process temperature [K]	\
0	1	M14860	M	298.1	308.6	
1	2	L47181	L	298.2	308.7	
2	3	L47182	L	298.1	308.5	
3	4	L47183	L	298.2	308.6	
4	5	L47184	L	298.2	308.7	

	Rotational speed [rpm]	Torque [Nm]	Tool wear [min]	Machine failure	T
0	1551	42.8	0	0	
1	1408	46.3	3	0	
2	1498	49.4	5	0	
3	1433	39.5	7	0	
4	1408	40.0	9	0	

	HDF	PWF	OSF	RNF
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0

Dataset info:

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 10000 entries, 0 to 9999
```

```
Data columns (total 14 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	Machine failure	10000 non-null	int64
9	TWF	10000 non-null	int64
10	HDF	10000 non-null	int64
11	PWF	10000 non-null	int64
12	OSF	10000 non-null	int64
13	RNF	10000 non-null	int64

```
dtypes: float64(3), int64(9), object(2)
```

```
memory usage: 1.1+ MB
```

```
None
```

```

# =====
# 2. Enhanced Exploratory Data Analysis (EDA)
# =====
print("\nSTEP 2: ENHANCED EXPLORATORY DATA ANALYSIS")
print("=====")

# Missing values analysis
print("\nMissing values:")
print(df.isnull().sum())

# Target distribution
plt.figure(figsize=(8, 5))
sns.countplot(x='Machine failure', data=df)
plt.title('Distribution of Machine Failures')
plt.show()

print("\nFailure distribution:")
print(df['Machine failure'].value_counts(normalize=True))

# Numerical features distribution
num_features = ['Air temperature [K]', 'Process temperature [K]',
                'Rotational speed [rpm]', 'Torque [Nm]', 'Tool wear [min]']

plt.figure(figsize=(15, 10))
for i, feature in enumerate(num_features, 1):
    plt.subplot(2, 3, i)
    sns.histplot(df[feature], kde=True)
    plt.title(f'Distribution of {feature}')
plt.tight_layout()
plt.show()

# Correlation analysis
plt.figure(figsize=(10, 8))
corr_matrix = df[num_features].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', center=0)
plt.title('Correlation Matrix of Numerical Features')
plt.show()

# Pairplot of numerical features with failure indication
sns.pairplot(df, vars=num_features, hue='Machine failure',
             palette={0: 'blue', 1: 'red'}, plot_kws={'alpha': 0.5})
plt.suptitle('Pairplot of Numerical Features Colored by Failure Status', y=1.02)
plt.show()

# Tool wear vs failure
plt.figure(figsize=(10, 6))
sns.boxplot(x='Machine failure', y='Tool wear [min]', data=df)
plt.title('Tool Wear Distribution by Failure Status')

```

```
plt.show()
```



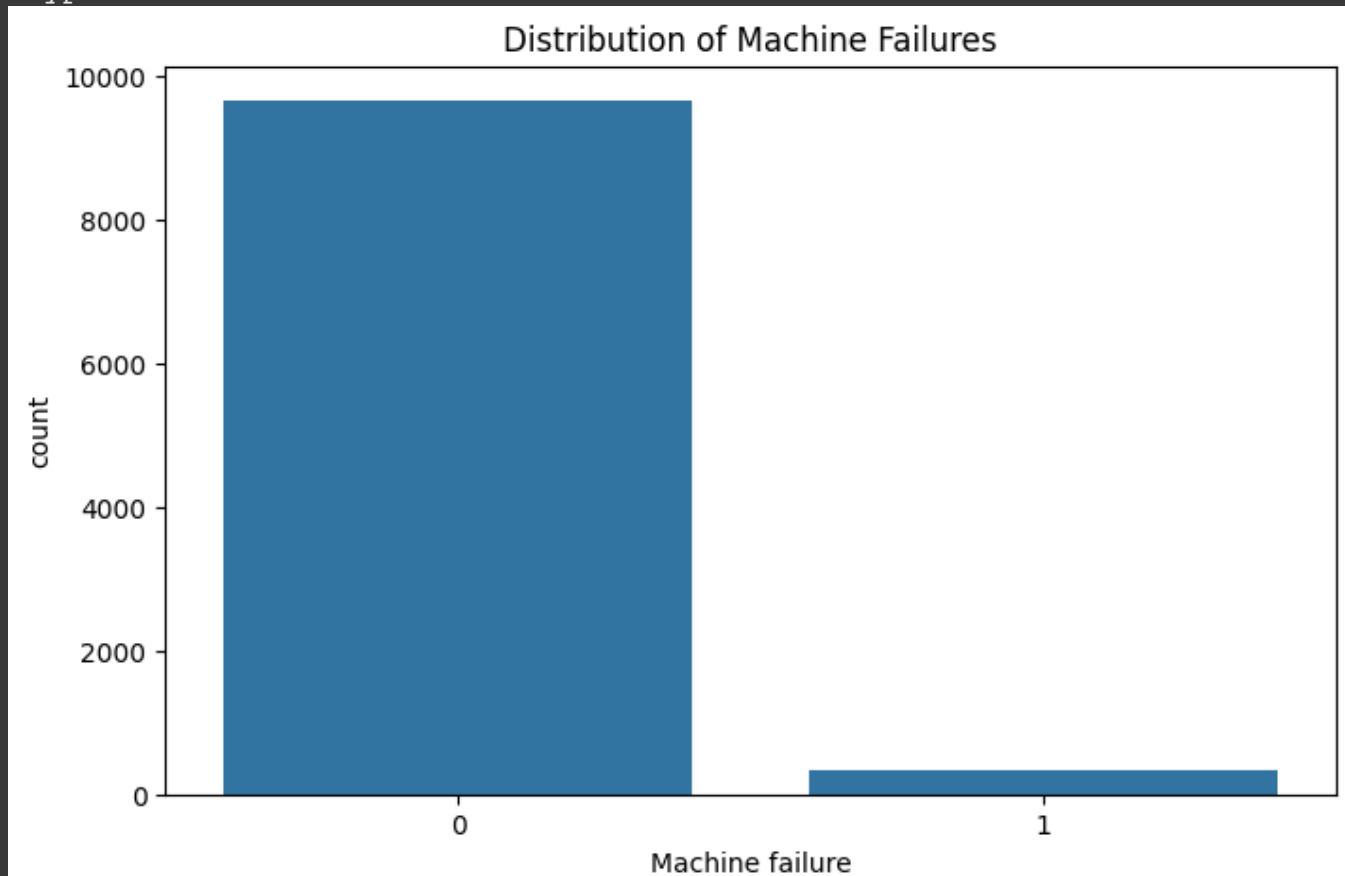
STEP 2: ENHANCED EXPLORATORY DATA ANALYSIS

=====

Missing values:

UDI	0
Product ID	0
Type	0
Air temperature [K]	0
Process temperature [K]	0
Rotational speed [rpm]	0
Torque [Nm]	0
Tool wear [min]	0
Machine failure	0
TWF	0
HDF	0
PWF	0
OSF	0
RNF	0

dtype: int64



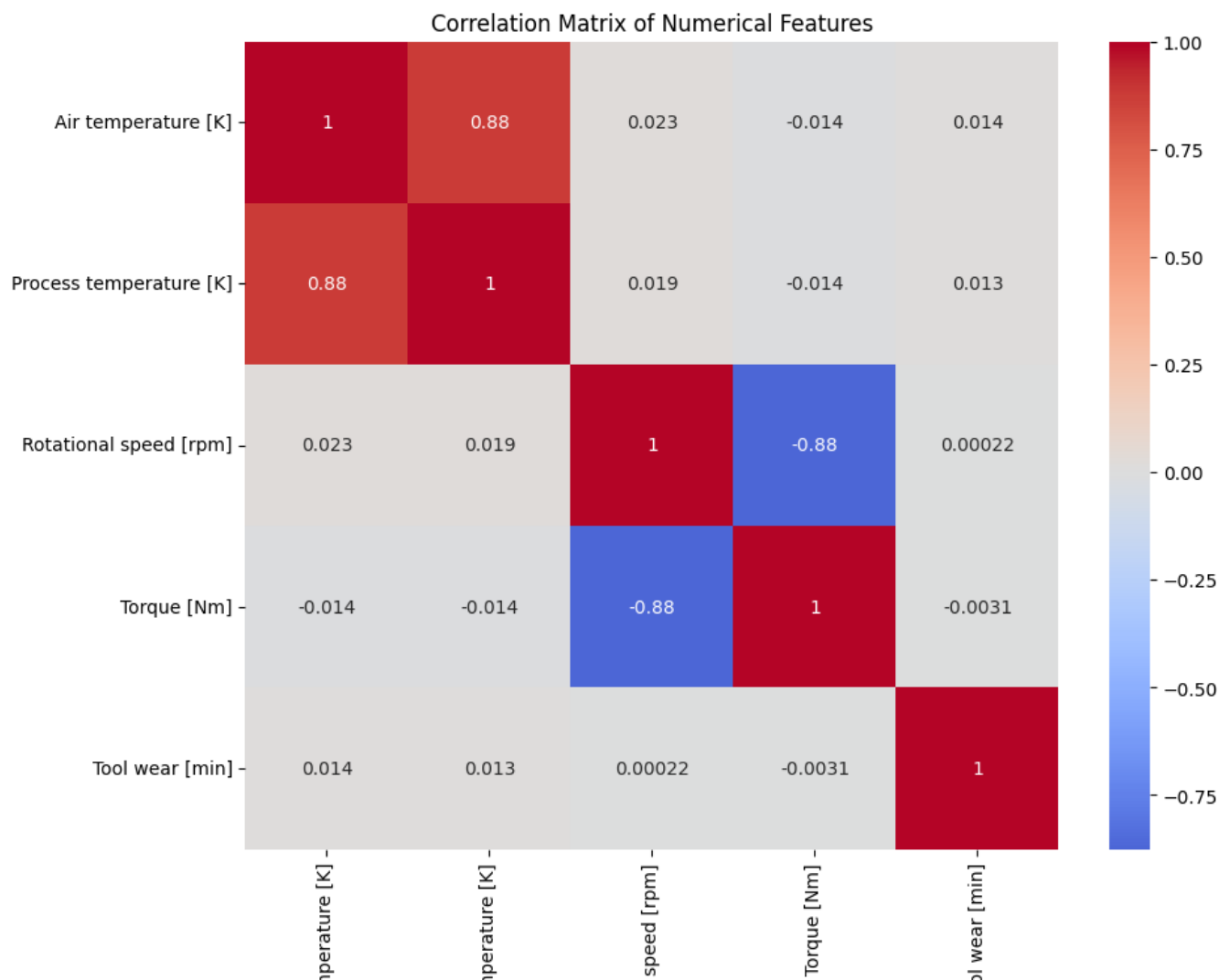
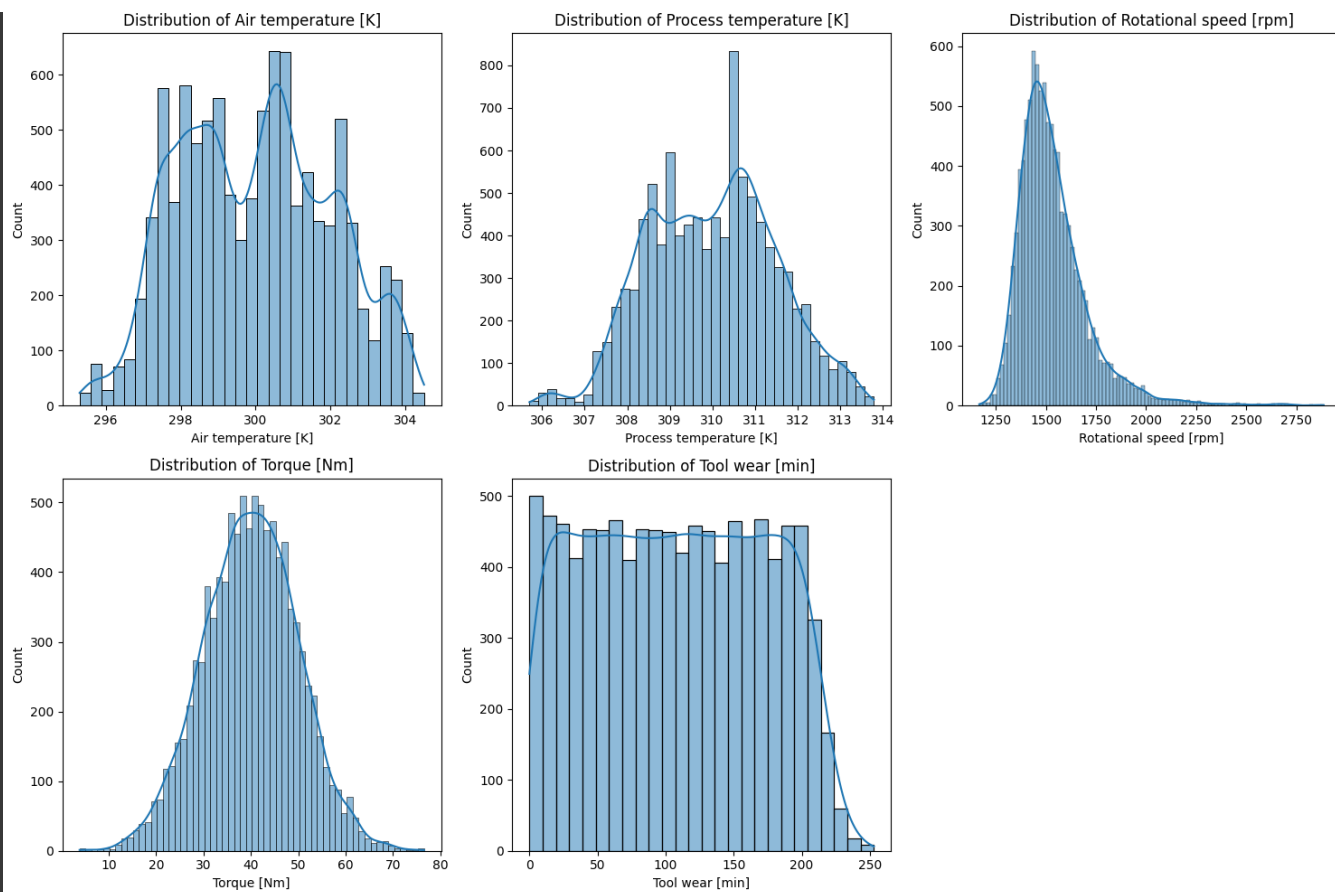
Failure distribution:

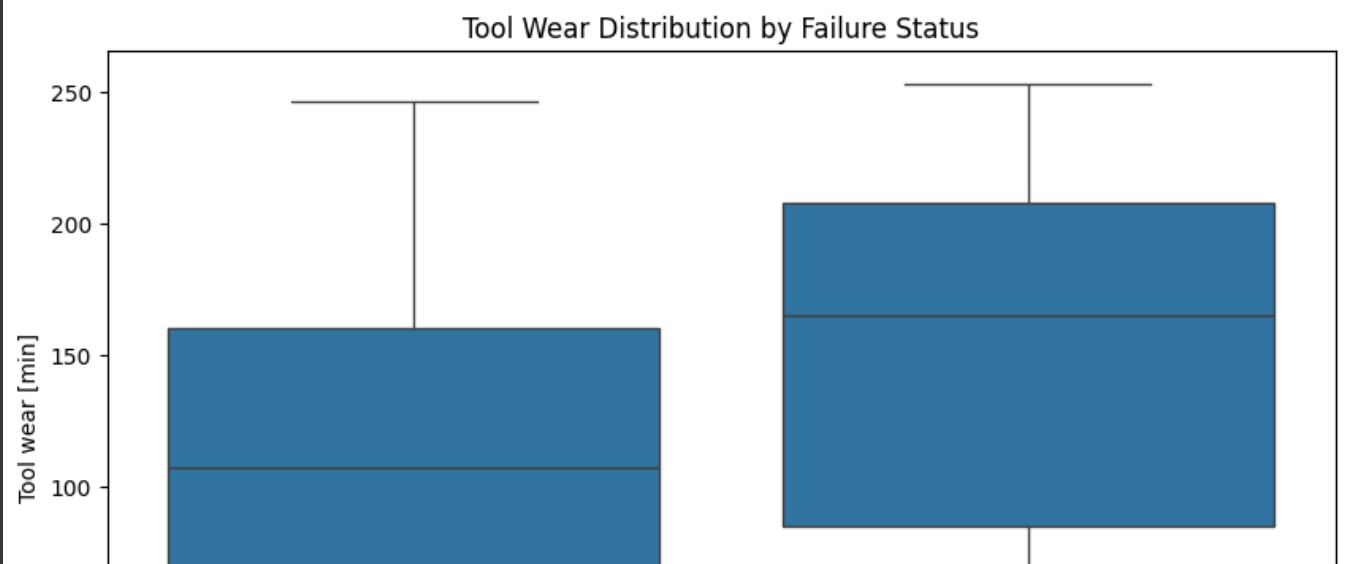
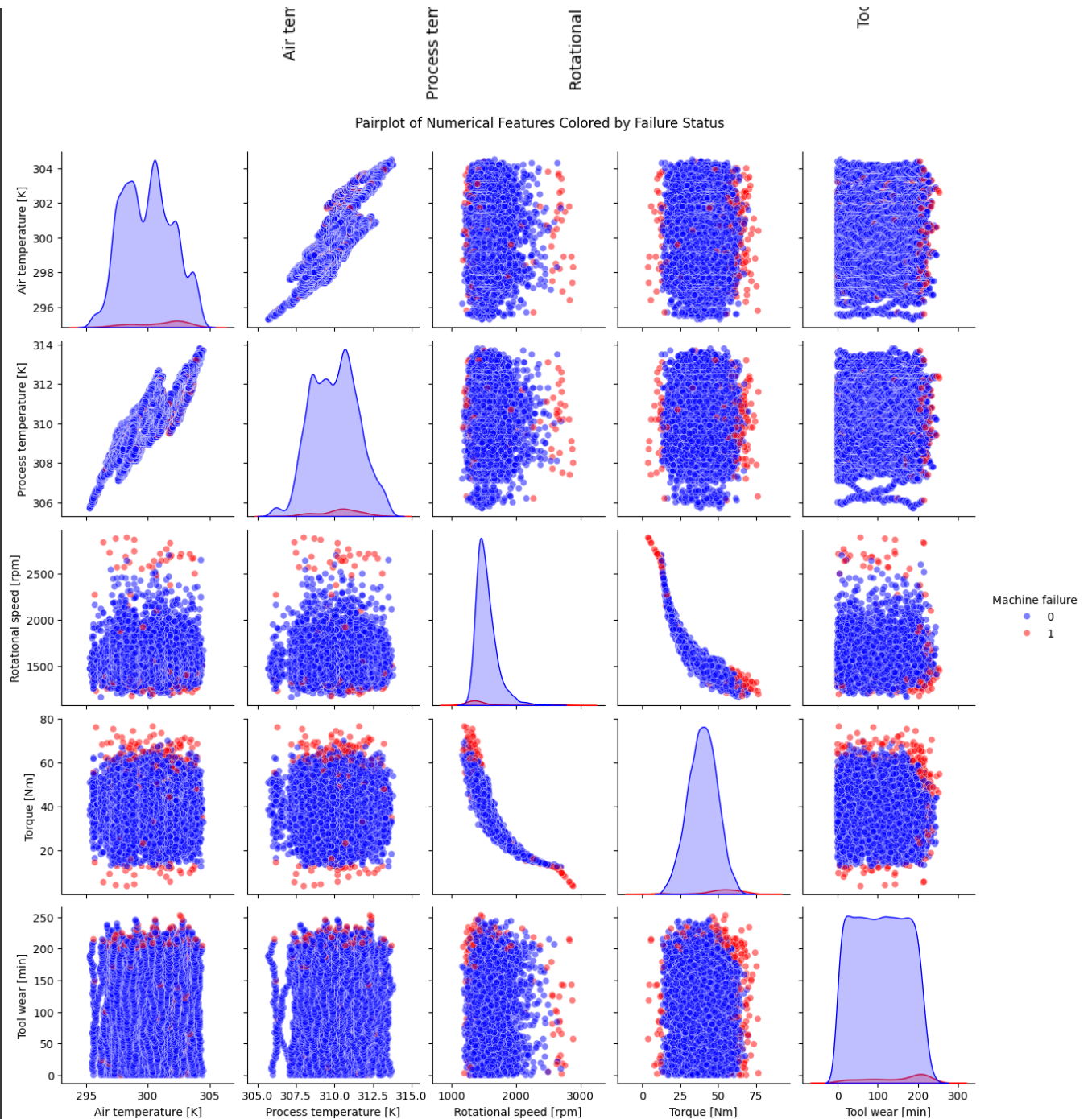
Machine failure

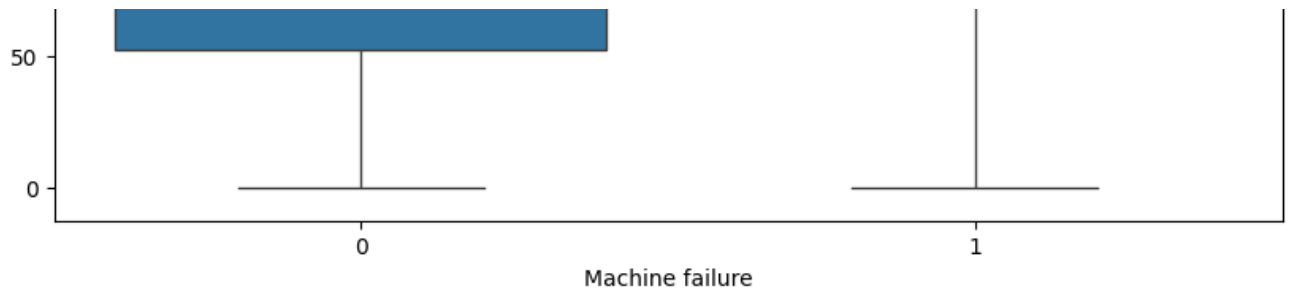
0 0.9661

1 0.0339

Name: proportion, dtype: float64








```
# =====
# 3. Feature Engineering
# =====
print("\nSTEP 3: FEATURE ENGINEERING")
print("=====")

def create_features(df):
    # Create power feature (Torque * Rotational speed)
    df['Power'] = df['Torque [Nm]'] * df['Rotational speed [rpm]']

    # Create temperature difference
    df['Temp_diff'] = df['Process temperature [K]'] - df['Air temperature [K]']

    # Create torque to speed ratio
    df['Torque_speed_ratio'] = df['Torque [Nm]'] / (df['Rotational speed [rpm]']

    # Create tool wear squared
    df['Tool_wear_squared'] = df['Tool wear [min]'] ** 2

    # Create overheating indicator
    df['Overheating'] = np.where(df['Process temperature [K]'] > 310, 1, 0)

    return df

df = create_features(df)

# Show new features
print("\nNew features created:")
print(df[['Power', 'Temp_diff', 'Torque_speed_ratio', 'Tool_wear_squared', 'Overheating']])
```



STEP 3: FEATURE ENGINEERING

=====

New features created:

	Power	Temp_diff	Torque_speed_ratio	Tool_wear_squared	Overheating
0	66382.8	10.5	0.027595	0	0
1	65190.4	10.5	0.032883	9	0
2	74001.2	10.4	0.032977	25	0
3	56603.5	10.4	0.027565	49	0
4	56320.0	10.5	0.028409	81	0

```

# =====
# 4. Data Preparation for Modeling
# =====
print("\nSTEP 4: DATA PREPARATION")
print("=====")

# Select features and target
features = [
    'Air temperature [K]',
    'Process temperature [K]',
    'Rotational speed [rpm]',
    'Torque [Nm]',
    'Tool wear [min]',
    'Power',
    'Temp_diff',
    'Torque_speed_ratio',
    'Tool_wear_squared',
    'Overheating'
]

target = 'Machine failure'

# Prepare data
X = df[features]
y = df[target]

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

print(f"\nTraining set shape: {X_train.shape}")
print(f"Test set shape: {X_test.shape}")
print(f"Class distribution in training set: {y_train.value_counts(normalize=True)}")

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

```



STEP 4: DATA PREPARATION

=====

```
Training set shape: (8000, 10)
Test set shape: (2000, 10)
Class distribution in training set: Machine failure
0      0.966125
1      0.033875
Name: proportion, dtype: float64
```

```
# =====
# 5. Machine Failure Prediction Model
# =====
print("\nSTEP 5: MACHINE FAILURE PREDICTION MODEL")
print("=====")

# Train XGBoost model (good for imbalanced data)
model = XGBClassifier(
    random_state=42,
    scale_pos_weight=(len(y_train) - sum(y_train)) / sum(y_train), # Handle cl
    eval_metric='logloss',
    n_estimators=200,
    max_depth=5,
    learning_rate=0.1
)

model.fit(X_train_scaled, y_train)

# Make predictions
y_pred = model.predict(X_test_scaled)
y_proba = model.predict_proba(X_test_scaled)[:, 1] # Probability of failure

# Evaluate model
print("\nModel Evaluation:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, y_proba)
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc
```

```
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()

# Feature importance
plt.figure(figsize=(12, 8))
sorted_idx = model.feature_importances_.argsort()
plt.barh(np.array(features)[sorted_idx], model.feature_importances_[sorted_idx])
plt.title('Feature Importance')
plt.show()
```



STEP 5: MACHINE FAILURE PREDICTION MODEL

=====

Model Evaluation:

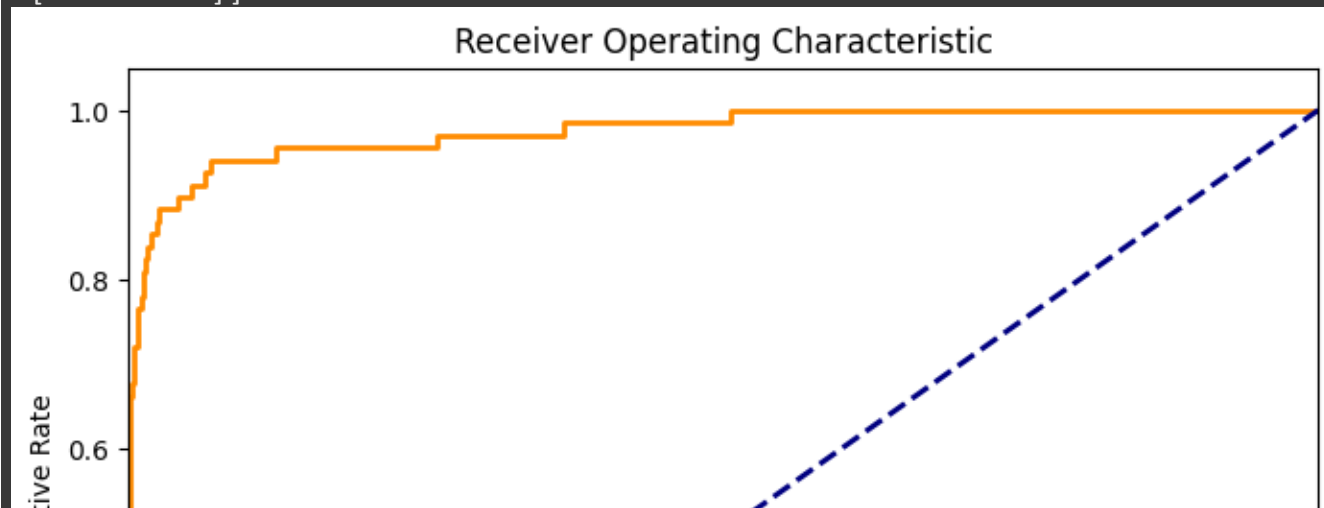
Accuracy: 0.9825

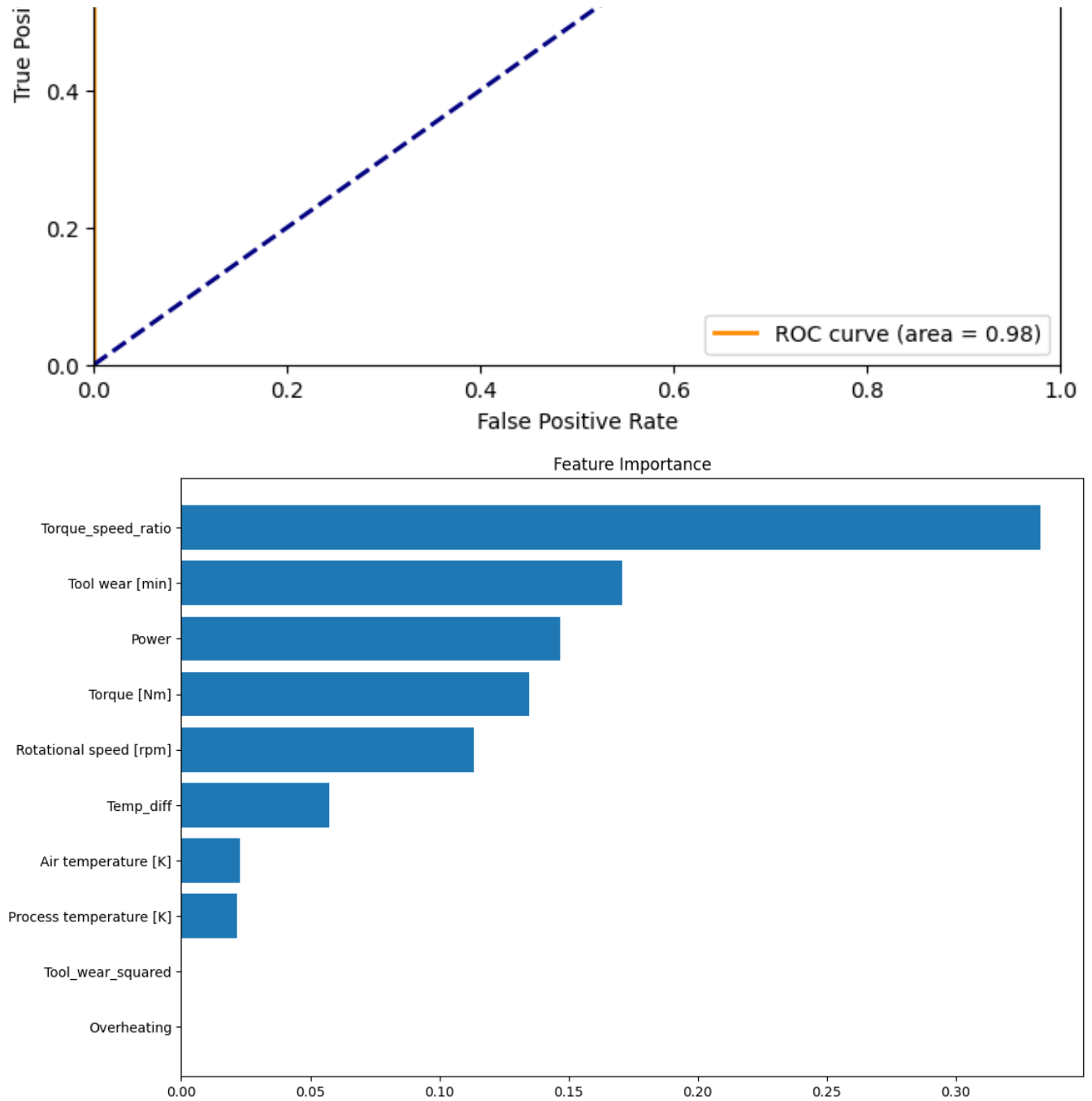
Classification Report:

	precision	recall	f1-score	support
0	0.99	0.99	0.99	1932
1	0.73	0.78	0.75	68
accuracy			0.98	2000
macro avg	0.86	0.88	0.87	2000
weighted avg	0.98	0.98	0.98	2000

Confusion Matrix:

```
[[1912  20]
 [  15  53]]
```





```
# =====
# 6. Anomaly Detection Implementation
# =====
print("\nSTEP 6: ANOMALY DETECTION")
print("=====")

# Initialize anomaly detection models
models = {
    "Isolation Forest": IsolationForest(n_estimators=150, contamination=0.05, r
    "Local Outlier Factor": LocalOutlierFactor(n_neighbors=25, contamination=0.
    "K-Nearest Neighbors (KNN)": KNN(contamination=0.05)
}

# Fit models and detect anomalies
anomaly_results = {}
for name, model_ad in models.items():
    if name == "Local Outlier Factor":
        anomalies = model_ad.fit_predict(X_train_scaled)
    else:
        model_ad.fit(X_train_scaled)
        anomalies = model_ad.predict(X_train_scaled)

# Convert predictions (1 = normal, -1 = anomaly)
anomalies = np.where(anomalies == 1, 0, 1)
anomaly_results[name] = anomalies

# Count anomalies
n_anomalies = sum(anomalies)
print(f"{name} detected {n_anomalies} anomalies ({n_anomalies/len(X_train_s

# Add anomaly labels to dataframe
train_indices = X_train.index
for name in models.keys():
    col_name = name.replace(" ", "_") + "_Anomaly"
    df.loc[train_indices, col_name] = anomaly_results[name]

# Visualize anomalies
def plot_anomalies(feature1, feature2):
    plt.figure(figsize=(10, 6))
    sns.scatterplot(data=df.loc[train_indices], x=feature1, y=feature2,
                    hue='Isolation_Forest_Anomaly', palette={0: 'blue', 1: 'red'})
    plt.title(f'Anomaly Detection: {feature1} vs {feature2}')
    plt.show()

# Plot some example feature pairs
plot_anomalies('Rotational speed [rpm]', 'Torque [Nm]')
plot_anomalies('Process temperature [K]', 'Air temperature [K]')
plot_anomalies('Tool wear [min]', 'Power')
```



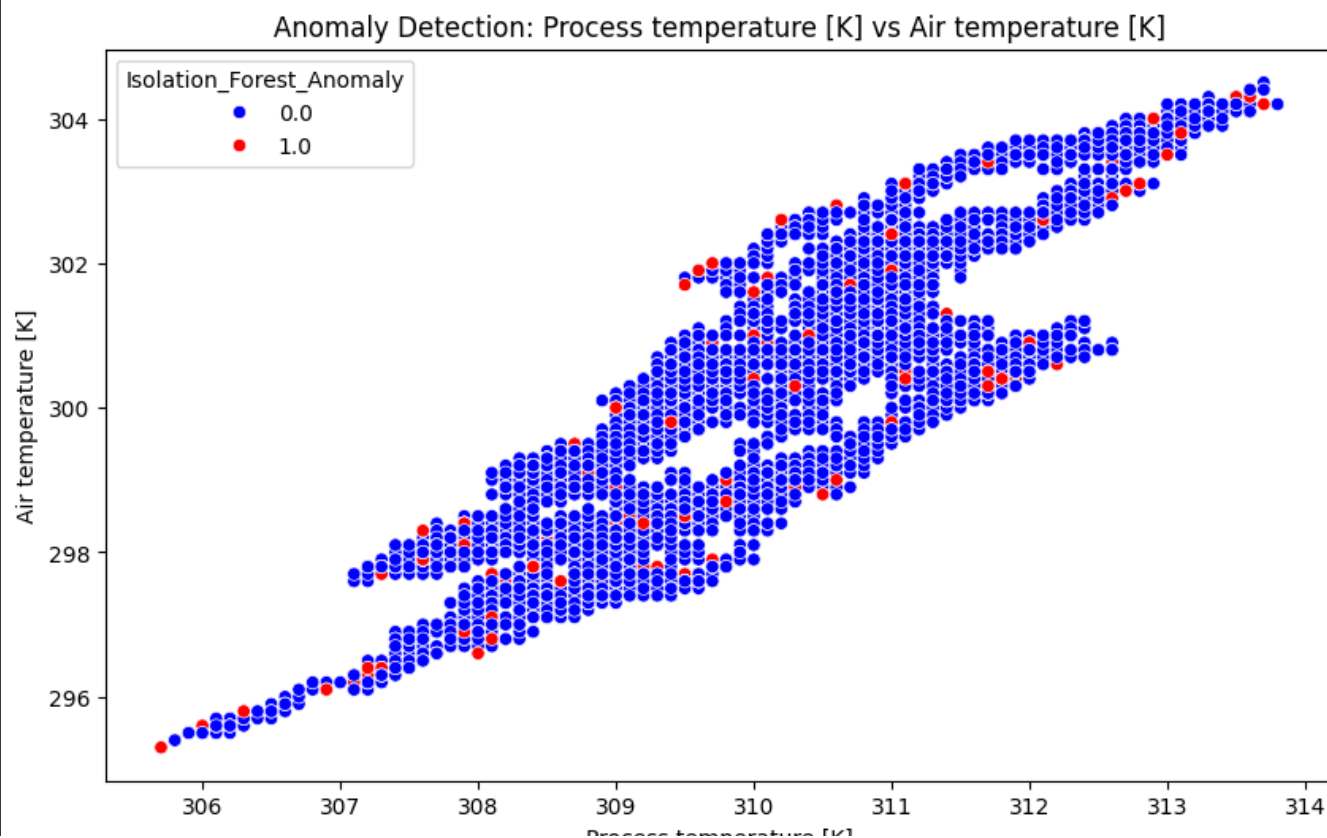
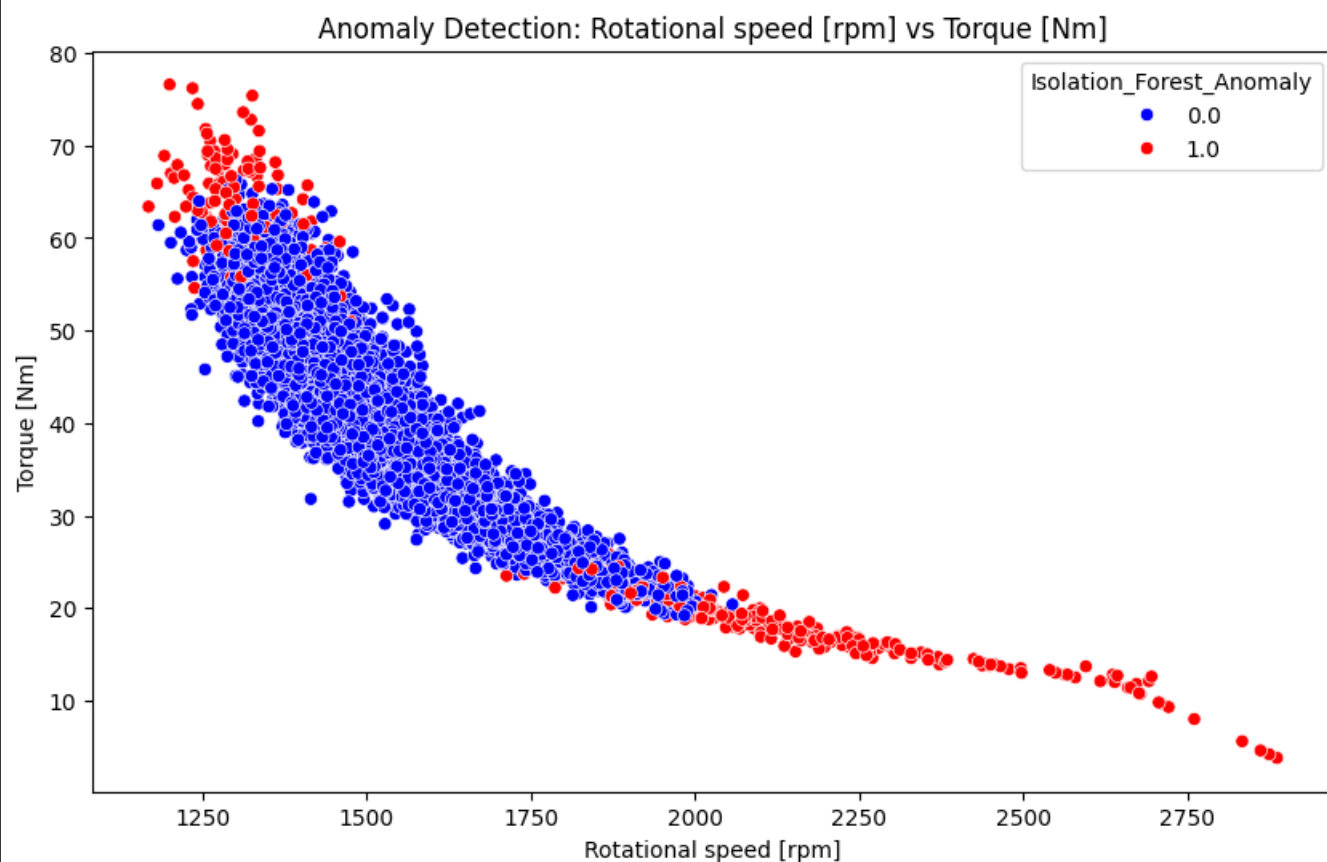
STEP 6: ANOMALY DETECTION

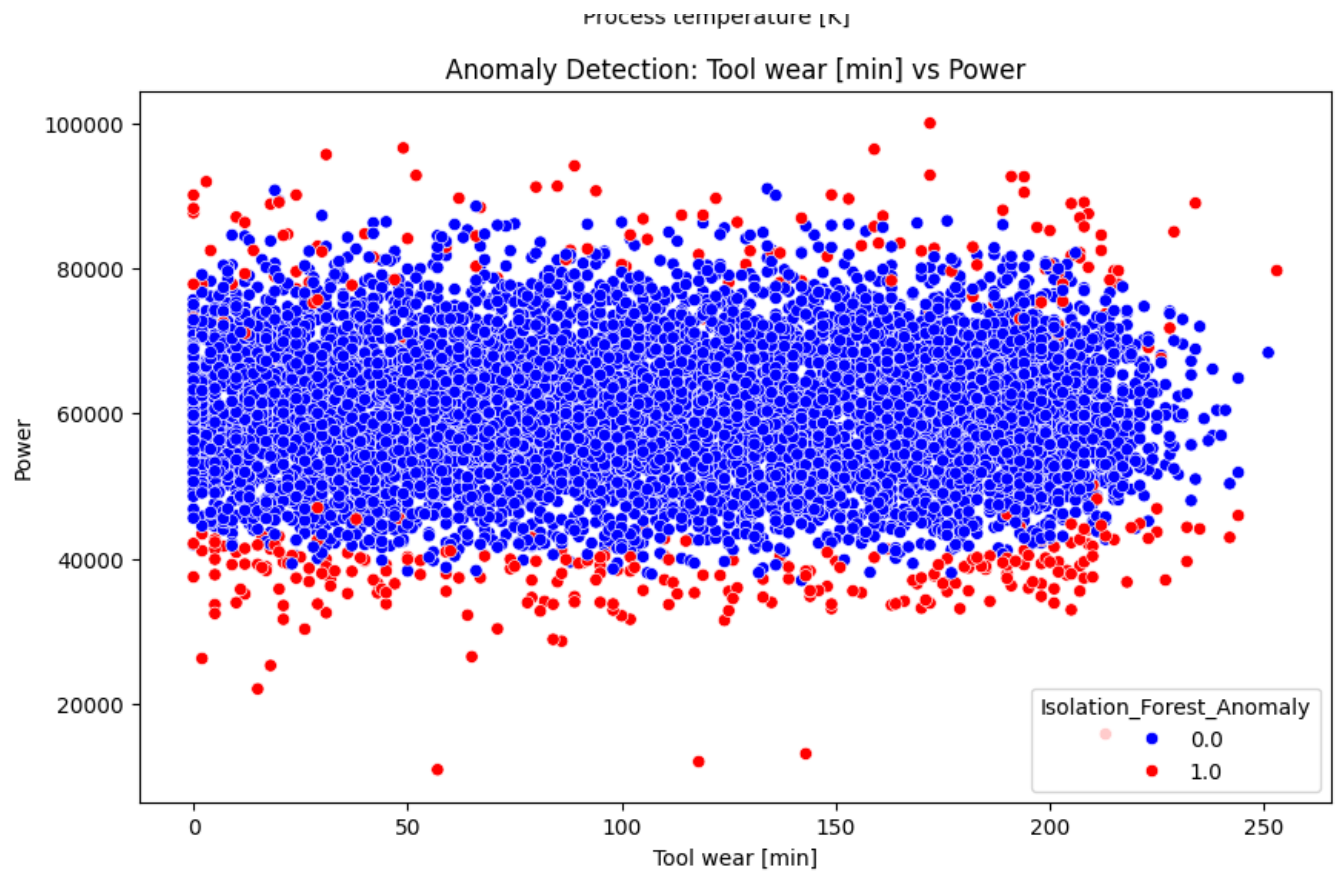
=====

Isolation Forest detected 400 anomalies (5.00% of training data)

Local Outlier Factor detected 400 anomalies (5.00% of training data)

K-Nearest Neighbors (KNN) detected 7701 anomalies (96.26% of training data)





=====


```

# 7. Risk Scoring System
# =====
print("\nSTEP 7: RISK SCORING SYSTEM")
print("=====")

# Create a function to calculate risk scores
def calculate_risk_score(row):
    score = 0

    # Get failure probability
    scaled_data = scaler.transform([row[features]])
    failure_prob = model.predict_proba(scaled_data)[0, 1]
    score += failure_prob * 50 # Weighted contribution

    # Add points for each anomaly detection method
    for name in models.keys():
        col_name = name.replace(" ", "_") + "_Anomaly"
        if col_name in row and not pd.isna(row[col_name]):
            score += row[col_name] * (20 if "Isolation" in name else 15)

    # Additional risk factors
    if row['Tool wear [min]'] > 200:
        score += 10
    if row['Overheating'] == 1:
        score += 15

    # Cap at 100
    return min(100, score)

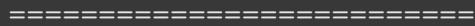
df['Risk_Score'] = df.apply(calculate_risk_score, axis=1)

# Visualize risk scores
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
sns.histplot(data=df, x='Risk_Score', bins=20, kde=True)
plt.title('Distribution of Risk Scores')

plt.subplot(1, 2, 2)
sns.boxplot(x='Machine failure', y='Risk_Score', data=df)
plt.title('Risk Scores by Failure Status')
plt.tight_layout()
plt.show()

# Analyze high-risk cases
high_risk = df[df['Risk_Score'] > 70]
print(f"\nFound {len(high_risk)} high-risk cases (Risk Score > 70):")
print(high_risk[features + ['Machine failure', 'Risk_Score']].describe())

```

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

```

/usr/local/lib/python2.11/dist-packages/cilconp/utilg/validation.py:2730: H

```

[illegible]

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255	256	257	258	259	260	261	262	263	264	265	266	267	268	269	270	271	272	273	274	275	276	277	278	279	280	281	282	283	284	285	286	287	288	289	290	291	292	293	294	295	296	297	298	299	300	301	302	303	304	305	306	307	308	309	310	311	312	313	314	315	316	317	318	319	320	321	322	323	324	325	326	327	328	329	330	331	332	333	334	335	336	337	338	339	340	341	342	343	344	345	346	347	348	349	350	351	352	353	354	355	356	357	358	359	360	361	362	363	364	365	366	367	368	369	370	371	372	373	374	375	376	377	378	379	380	381	382	383	384	385	386	387	388	389	390	391	392	393	394	395	396	397	398	399	400	401	402	403	404	405	406	407	408	409	410	411	412	413	414	415	416	417	418	419	420	421	422	423	424	425	426	427	428	429	430	431	432	433	434	435	436	437	438	439	440	441	442	443	444	445	446	447	448	449	450	451	452	453	454	455	456	457	458	459	460	461	462	463	464	465	466	467	468	469	470	471	472	473	474	475	476	477	478	479	480	481	482	483	484	485	486	487	488	489	490	491	492	493	494	495	496	497	498	499	500	501	502	503	504	505	506	507	508	509	510	511	512	513	514	515	516	517	518	519	520	521	522	523	524
---	---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

1	13	3	3/11	1	0.44/31	1	13	13	3/11	1	0.500
---	----	---	------	---	---------	---	----	----	------	---	-------

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

```
/usr/local/lib/ruby2.1.1/dist/packages/glibc-utils/validation: u
```

[illegible]

1	13	3	3/11	1	11	0.44	31	1	1	13	1	13	1	3	11	1	0.500
---	----	---	------	---	----	------	----	---	---	----	---	----	---	---	----	---	-------

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

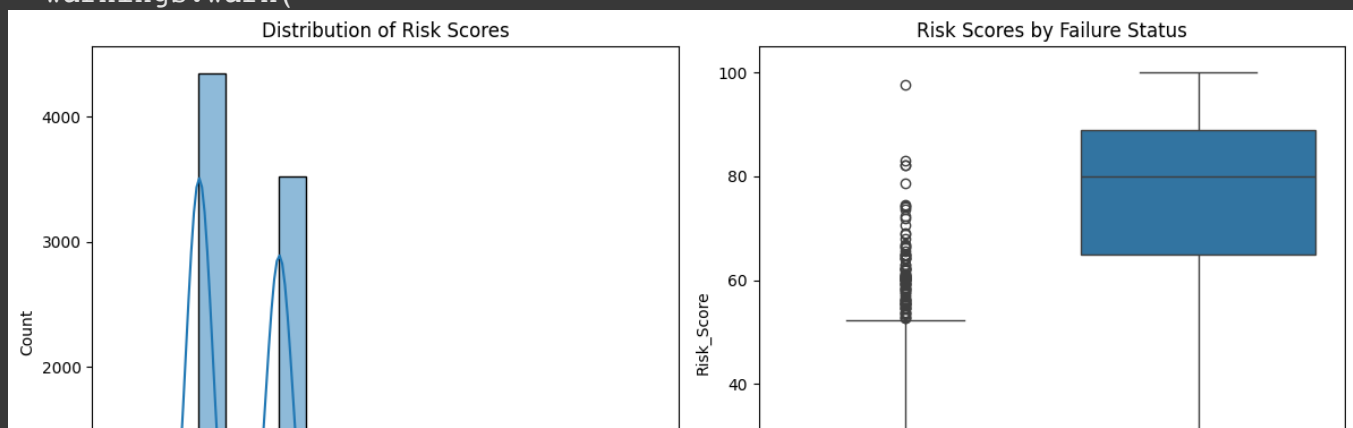
[illegible]

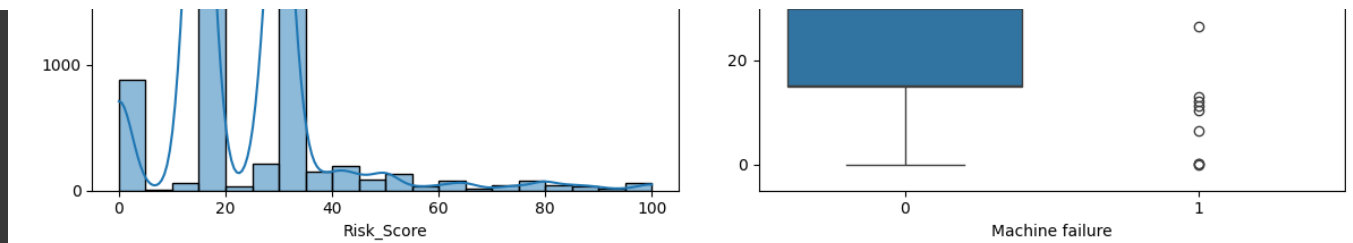
[illegible]

```

/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: U
warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: U
warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: U
warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: U
warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: U
warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: U
warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: U
warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: U
warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: U
warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: U
warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: U
warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: U
warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: U
warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: U
warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: U
warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: U
warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: U
warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: U
warnings.warn(

```





Found 260 high-risk cases (Risk Score > 70):

	Air temperature [K]	Process temperature [K]	Rotational speed [rpm]
count	260.000000	260.000000	260.000000
mean	301.120769	310.505385	1519.219231
std	2.013015	1.332447	415.023275
min	295.700000	306.200000	1181.000000
25%	299.600000	309.975000	1326.000000
50%	301.800000	310.600000	1366.000000
75%	302.600000	311.400000	1442.000000
max	304.400000	313.700000	2886.000000

	Torque [Nm]	Tool wear [min]	Power	Temp_diff \
count	260.000000	260.000000	260.000000	260.000000
mean	49.049615	149.242308	68232.337692	9.384615
std	16.978932	74.816938	18105.814503	1.158115
min	3.800000	0.000000	10966.800000	7.600000
25%	45.375000	84.750000	62945.350000	8.400000
50%	52.650000	182.500000	71359.200000	9.200000
75%	60.150000	210.000000	79923.500000	10.400000
max	76.600000	253.000000	99980.400000	12.000000

	Torque_speed_ratio	Tool_wear_squared	Overheating	Machine failure
count	260.000000	260.000000	260.000000	260.000000
mean	0.035772	27849.311538	0.746154	0.953846
std	0.014486	19210.118099	0.436050	0.210223
min	0.001317	0.000000	0.000000	0.000000
25%	0.032983	7182.750000	0.000000	1.000000
50%	0.038155	33306.500000	1.000000	1.000000
75%	0.044665	44100.000000	1.000000	1.000000
max	0.063833	64009.000000	1.000000	1.000000

	Risk_Score
count	260.000000
mean	85.823738
std	9.085919
min	70.585716
25%	79.816120
50%	84.709633
75%	94.844568
max	100.000000

```

# =====
# 8. Model Testing with Random Sample
# =====
print("\nSTEP 8: TESTING WITH RANDOM SAMPLE")
print("=====")

def evaluate_sample(sample, actual, X_train_scaled):
    """Evaluate a single sample with proper handling for anomaly detection"""
    try:
        # Prepare the sample
        sample_df = pd.DataFrame([sample])
        sample_df = create_features(sample_df)
        scaled_sample = scaler.transform(sample_df[features])

        # Get failure prediction
        failure_prob = model.predict_proba(scaled_sample)[0, 1]
        failure_pred = model.predict(scaled_sample)[0]

        # Initialize anomaly results
        anomaly_results = {}
        anomaly_scores = {}

        # Handle each anomaly detection model differently
        for name, model_ad in models.items():
            if name == "Local Outlier Factor":
                # For LOF, we need to include training data
                combined_data = np.vstack([X_train_scaled, scaled_sample])
                n_neighbors = min(25, len(X_train_scaled) - 1)
                lof = LocalOutlierFactor(n_neighbors=n_neighbors, contamination=0.1)
                lof.fit(combined_data)
                anomaly_results[name] = lof.fit_predict(combined_data)[-1] # 1
                anomaly_scores[name] = lof.negative_outlier_factor_[-1]
            else:
                # Other models can handle single samples
                model_ad.fit(X_train_scaled) # Refit on training data
                anomaly_results[name] = model_ad.predict(scaled_sample)[0]
                if hasattr(model_ad, 'decision_function'):
                    anomaly_scores[name] = model_ad.decision_function(scaled_sample)
                else:
                    anomaly_scores[name] = np.nan

        # Calculate risk score
        risk_score = calculate_risk_score(sample_df.iloc[0])

```



```

# Prepare results
results = {
    'sample_index': random_idx,
    'actual_failure': bool(actual),
    'features': sample.to_dict(),
    'predictions': {
        'failure_probability': float(failure_prob),
        'predicted_failure': bool(failure_pred),
        'anomaly_detection': {
            name: {
                'result': 'Anomaly' if result == -1 else 'Normal',
                'score': float(score)
            } for name, (result, score) in zip(anomaly_results.keys(),
                                             zip(anomaly_results.values(),
                                                 anomaly_scores.values()))
        }
    },
    'risk_assessment': {
        'risk_score': float(risk_score),
        'risk_level': 'High' if risk_score > 70 else ('Medium' if risk_
    }
}
return results

```

```

except Exception as e:
    print(f"Error evaluating sample: {str(e)}")
    return None

```

```

# Select and evaluate a random sample
random_idx = random.choice(X_test.index)
random_sample = X_test.loc[random_idx]
actual_failure = y_test.loc[random_idx]

```

```

# Evaluate with training data available for anomaly detection
sample_result = evaluate_sample(random_sample, actual_failure, X_train_scaled)

```

```

# Print results
if sample_result:
    print("\n" + "="*50)
    print(f"RANDOM SAMPLE EVALUATION (Index: {sample_result['sample_index']})")
    print("="*50)

    print(f"\n{'Actual Failure:':<25} {'Yes' if sample_result['actual_failure']")
    print(f"{'Predicted Failure:':<25} {'Yes' if sample_result['predictions']['")
    print(f"{'Failure Probability:':<25} {sample_result['predictions']['failure")

    print("\nANOMALY DETECTION RESULTS:")
    for algo, data in sample_result['predictions']['anomaly_detection'].items()

```

```

        print(f"{algo:<25} {data['result']} (Score: {data['score']:.2f})")

    print("\nRISK ASSESSMENT:")
    print(f"{'Risk Score:':<25} {sample_result['risk_assessment']['risk_score']}")
    print(f"{'Risk Level:':<25} {sample_result['risk_assessment']['risk_level']}")

    print("\nKEY FEATURE VALUES:")
    features_to_show = ['Tool wear [min]', 'Power', 'Process temperature [K]',
                        'Rotational speed [rpm]', 'Torque [Nm]']
    for feature in features_to_show:
        print(f"{feature:<25} {sample_result['features'][feature]:.2f}")

    print("\n" + "="*50)
else:
    print("Failed to evaluate sample.")

```



STEP 8: TESTING WITH RANDOM SAMPLE

=====

=====

RANDOM SAMPLE EVALUATION (Index: 8967)

=====

Actual Failure:	No
Predicted Failure:	No
Failure Probability:	0.0002

ANOMALY DETECTION RESULTS:

Isolation Forest	Normal (Score: 0.13)
Local Outlier Factor	Normal (Score: -1.05)
K-Nearest Neighbors (KNN)	Normal (Score: 0.61)

RISK ASSESSMENT:

Risk Score:	0.0
Risk Level:	Low

KEY FEATURE VALUES:

Tool wear [min]	92.00
Power	62054.40
Process temperature [K]	307.80
Rotational speed [rpm]	1536.00
Torque [Nm]	40.40

=====

/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: U

warnings.warn(

```
# =====
# 9. Model Saving
# =====
print("\nSTEP 9: SAVING MODELS")
print("=====")

# Save models
joblib.dump(model, 'failure_prediction_model.pkl')
joblib.dump(scaler, 'scaler.pkl')
for name, model_ad in models.items():
    filename = name.lower().replace(" ", "_") + "_anomaly.pkl"
    joblib.dump(model_ad, filename)

# Save the enhanced dataframe
df.to_csv('enhanced_predictive_maintenance.csv', index=False)

print("\nAll models and enhanced dataset saved successfully!")
```



```
STEP 9: SAVING MODELS
=====
```

```
All models and enhanced dataset saved successfully!
```

```
!pip install -q -U google-generativeai flask_ngrok
```

```
!pip install -q -U google-generativeai fastapi nest-asyncio pyngrok uvicorn
```

```
# =====
# Predictive Maintenance Chat API with FastAPI
# =====
!pip install -q -U google-generativeai fastapi nest-asyncio pyngrok uvicorn
from fastapi import FastAPI, HTTPException
from fastapi.middleware.cors import CORSMiddleware
import uvicorn
from pyngrok import ngrok
import nest_asyncio
import google.generativeai as genai
from pydantic import BaseModel
import json
import pandas as pd
import random
import threading

# Configure Gemini
```

```

GOOGLE_API_KEY = "AIzaSyBgwZWa8WkwfV1HPiR-P86GpkrCF4IEam4" # Replace with your
genai.configure(api_key=GOOGLE_API_KEY)

# Initialize FastAPI app
app = FastAPI(title="Predictive Maintenance Chat API")

# CORS configuration
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# =====
# Agent Definitions
# =====

class PredictiveAgent:
    def __init__(self):
        self.model = genai.GenerativeModel('gemini-1.5-flash')
        self.name = "Predictive Analyst"
        self.instruction = """Analyze equipment sensor data to predict failures.
Respond with:
- failure_probability: 0-1
- risk_factors: list
- immediate_actions: list"""

    async def analyze(self, sensor_data: dict):
        prompt = f"""Analyze this industrial equipment data:
{json.dumps(sensor_data, indent=2)}

Provide:
1. Probability of failure (0-1)
2. Top 3 risk factors
3. 2 recommended immediate actions"""

        response = await self.model.generate_content_async(prompt)
        return {
            "agent": self.name,
            "response": response.text,
            "type": "analysis"
        }

class GuidanceAgent:
    def __init__(self):
        self.model = genai.GenerativeModel('gemini-1.5-flash')
        self.name = "Maintenance Advisor"

```

```

self.name = "Maintenance Advisor"
self.instruction = """Create maintenance plans based on technical analysis
Respond with:
- priority: Critical/High/Medium/Low
- tasks: list
- required_parts: list
- estimated_downtime: hours"""

```

```

async def advise(self, analysis: str):
    prompt = f"""Create maintenance plan for this analysis:
    {analysis}

    Include:
    1. Priority level
    2. Specific maintenance tasks
    3. Required parts/tools
    4. Estimated downtime in hours"""

    response = await self.model.generate_content_async(prompt)
    return {
        "agent": self.name,
        "response": response.text,
        "type": "guidance"
    }

```

```

class RemedyAgent:
    def __init__(self):
        self.model = genai.GenerativeModel('gemini-1.5-flash')
        self.name = "Repair Specialist"
        self.instruction = """Provide step-by-step repair procedures.
Respond with:
- steps: numbered list
- safety_checks: list
- tools_required: list
- time_per_step: minutes"""

    async def create_procedure(self, guidance: str):
        prompt = f"""Create repair procedure for:
        {guidance}

        For each step include:
        1. Tools/parts needed
        2. Detailed instructions
        3. Safety checks
        4. Time estimate in minutes"""

        response = await self.model.generate_content_async(prompt)
        return {
            "agent": self.name,

```

```

        "response": response.text,
        "type": "remedy"
    }

# Initialize agents
predictive_agent = PredictiveAgent()
guidance_agent = GuidanceAgent()
remedy_agent = RemedyAgent()

# =====
# Request/Response Models
# =====

class ChatRequest(BaseModel):
    message: str
    session_id: str = "default"

class ChatResponse(BaseModel):
    session_id: str
    agent: str
    response: str
    response_type: str

# =====
# API Endpoints
# =====

@app.post("/chat", response_model=ChatResponse)
async def chat_endpoint(request: ChatRequest):
    try:
        # Generate realistic sensor data
        sensor_data = {
            "air_temp_k": round(random.uniform(295, 310), 2),
            "process_temp_k": round(random.uniform(305, 320), 2),
            "rotational_speed_rpm": random.randint(1000, 3000),
            "torque_nm": round(random.uniform(30, 70), 2),
            "tool_wear_min": random.randint(0, 250)
        }

        user_input = request.message.lower()

        if "analyze" in user_input:
            response = await predictive_agent.analyze(sensor_data)
        elif "maintenance" in user_input or "plan" in user_input:
            analysis = await predictive_agent.analyze(sensor_data)
            response = await guidance_agent.advise(analysis["response"])
        elif "repair" in user_input or "procedure" in user_input:
            analysis = await predictive_agent.analyze(sensor_data)

```

```

        guidance = await guidance_agent.advise(analysis["response"])
        response = await remedy_agent.create_procedure(guidance["response"])
    else:
        response = {
            "agent": "System",
            "response": "Please specify what you need:\n- 'Analyze equipment
            "type": "info"
        }

    return {
        "session_id": request.session_id,
        "agent": response["agent"],
        "response": response["response"],
        "response_type": response["type"]
    }

except Exception as e:
    raise HTTPException(status_code=500, detail=str(e))

@app.get("/sensor_data")
async def get_sensor_data():
    """Endpoint to get sample sensor data"""
    return {
        "air_temp_k": round(random.uniform(295, 310), 2),
        "process_temp_k": round(random.uniform(305, 320), 2),
        "rotational_speed_rpm": random.randint(1000, 3000),
        "torque_nm": round(random.uniform(30, 70), 2),
        "tool_wear_min": random.randint(0, 250)
    }

# =====
# Server Setup
# =====

def run_server():
    # Allow nested asyncio
    nest_asyncio.apply()

    # Start ngrok tunnel
    ngrok_tunnel = ngrok.connect(8000)
    print(f"Public URL: {ngrok_tunnel.public_url}")

    # Start FastAPI server
    uvicorn.run(app, host="0.0.0.0", port=8000)

# Start the server in a separate thread
threading.Thread(target=run_server, daemon=True).start()

```

```
print("""
=== Predictive Maintenance Chat API ===
Your FastAPI server is now running!

Available endpoints:
- POST /chat - Chat with the agents
- GET /sensor_data - Get sample sensor data

Try these commands:
1. "Analyze current equipment status"
2. "Suggest maintenance plan"
3. "Provide repair procedure"

The public URL will appear above when ngrok initializes.
""")
```



```
=== Predictive Maintenance Chat API ===
Your FastAPI server is now running!

Available endpoints:
- POST /chat - Chat with the agents
- GET /sensor_data - Get sample sensor data

Try these commands:
1. "Analyze current equipment status"
2. "Suggest maintenance plan"
3. "Provide repair procedure"

The public URL will appear above when ngrok initializes.
```

```
# =====
# 11 extra. First run this setup cell
# =====
!pip install -q -U google-generativeai fastapi nest-asyncio pyngrok uvicorn
!ngrok authtoken YOUR_NGROK_AUTH_TOKEN # REPLACE WITH YOUR TOKEN

# =====
# 2. Then run this main cell
# =====
from fastapi import FastAPI, Request, HTTPException
from fastapi.responses import HTMLResponse
import uvicorn
from pyngrok import ngrok
import nest_asyncio
import google.generativeai as genai
import json
import random
import threading
```



```

import threading
from typing import Dict

# Configure Gemini
GOOGLE_API_KEY = "YOUR_API_KEY" # REPLACE WITH YOUR KEY
genai.configure(api_key=GOOGLE_API_KEY)

# Initialize FastAPI
app = FastAPI()

# Agent Definitions
class PredictiveAgent:
    def __init__(self):
        self.model = genai.GenerativeModel('gemini-1.5-flash')
    async def analyze(self, sensor_data: Dict) -> Dict:
        response = await self.model.generate_content_async(
            f"Analyze this equipment data:\n{json.dumps(sensor_data, indent=2)}"
        )
        return {"agent": "Analyst", "response": response.text}

@app.post("/api/chat")
async def chat_endpoint(request: Dict):
    sensor_data = {
        "air_temp": round(random.uniform(295, 310), 2),
        "process_temp": round(random.uniform(305, 320), 2),
        "rpm": random.randint(1000, 3000),
        "torque": round(random.uniform(30, 70), 2),
        "tool_wear": random.randint(0, 250)
    }
    response = await PredictiveAgent().analyze(sensor_data)
    return response

@app.get("/", response_class=HTMLResponse)
async def chat_interface(request: Request):
    return """
<html>
<body>
    <h1>Predictive Maintenance Chat</h1>
    <div id="chat"></div>
    <input type="text" id="message">
    <button onclick="sendMessage()">Send</button>
    <script>
        async function sendMessage() {
            const response = await fetch('/api/chat', {
                method: 'POST',
                headers: {'Content-Type': 'application/json'},
                body: JSON.stringify({message: document.getElementById('mess
            });
            const data = await response.json();

```

```


        document.getElementById('chat').innerHTML +=
            `<p><b>${data.agent}</b> ${data.response}</p>`;
    }
</script>
</body>
</html>
"""

# Start server with explicit URL printing
def run_server():
    nest_asyncio.apply()
    ngrok_tunnel = ngrok.connect(8000)
    print("\n=== Chat Interface Ready ===")
    print(f"Access at: {ngrok_tunnel.public_url}")
    print("\nIf the URL doesn't appear, check the ngrok dashboard:")
    print(f"https://dashboard.ngrok.com/status/tunnels")
    uvicorn.run(app, host="0.0.0.0", port=8000)

# Keep Colab alive
from IPython.display import Javascript
Javascript("""
function keepAlive() {
    console.log("Keeping colab alive");
    document.querySelector("colab-toolbar-button#connect").click()
}
setInterval(keepAlive, 60000)
""")

# Start server
threading.Thread(target=run_server, daemon=True).start()

```

 Authtoken saved to configuration file: /root/.config/ngrok/ngrok.yml

