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

# Import libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
from xgboost import XGBClassifier
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import roc_curve, auc, precision_recall_curve

# Set style for plots
sns.set_style('whitegrid')
plt.rcParams['figure.figsize'] = (12, 6)
# Load the dataset
from google.colab import files
uploaded = files.upload()

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

# Basic data exploration
print("\nMissing values:")
print(df.isnull().sum())

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

# Feature engineering
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]'] + 0.001)

    return df

df = create_features(df)

# 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'
]

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)

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

# 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 class imbalanc
    eval_metric='logloss'
)

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))

# Feature importance
plt.figure(figsize=(10, 6))
plt.barh(features, model.feature_importances_)
plt.title('Feature Importance')
plt.show()

# Example prediction
sample_data = X_test_scaled[0:10]  # Take first test sample
sample_pred = model.predict(sample_data)
sample_prob = model.predict_proba(sample_data)[0, 1]

print("\nExample Prediction:")
print("Features:", X_test.iloc[10].to_dict())
print(f"Predicted failure: {'Yes' if sample_pred[0] else 'No'} (Probability: {sample_prob:.
print("Actual failure:", y_test.iloc[0])

# Save model for later use
```

```
import joblib
joblib.dump(model, 'predictive_maintenance_model.pkl')
joblib.dump(scaler, 'scaler.pkl')

print("\nModel and scaler saved to disk.")
```

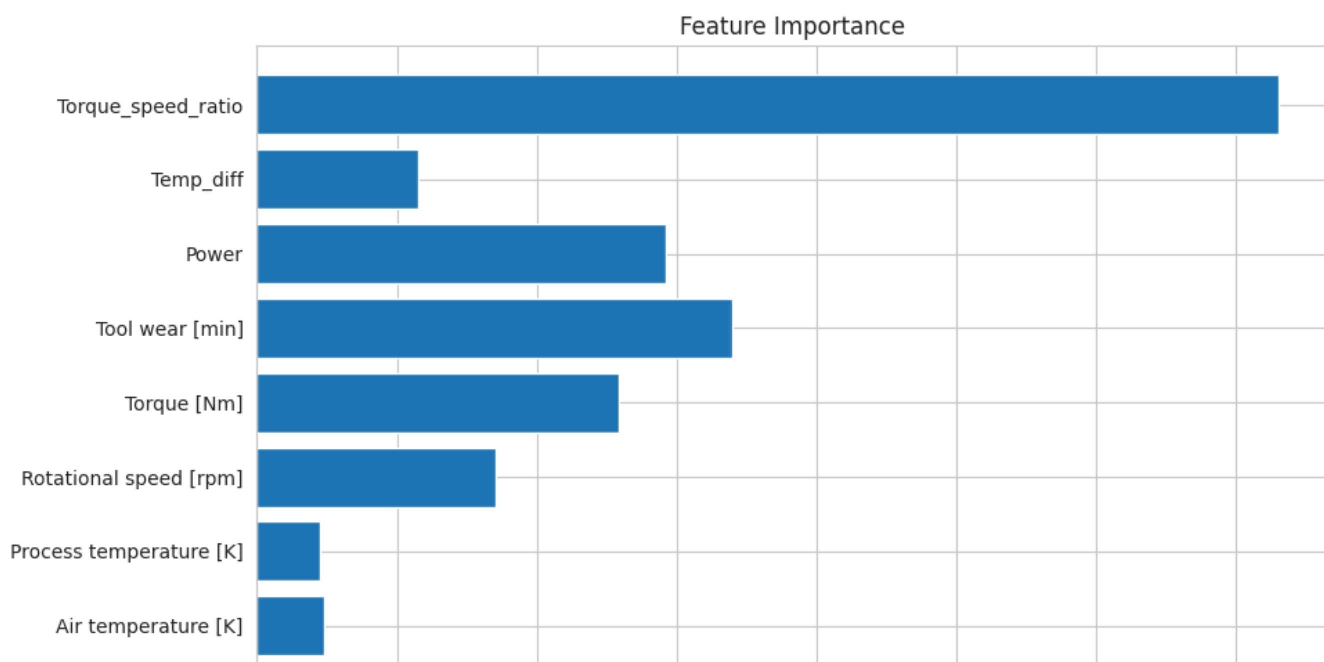
```
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: xgboost in /usr/local/lib/python3.11/dist-package
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: numpy>=1.23.2 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.11/dis
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dis
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/di
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/di
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dis
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packag
```

Choose files ai4i2020.csv

- **ai4i2020.csv**(text/csv) - 522048 bytes, last modified: 12/06/2025 - 100% done

```
Saving ai4i2020.csv to ai4i2020 (6).csv
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  TWF  \
0                    1551         42.8                0                0    0
1                    1408         46.3                3                0    0
2                    1498         49.4                5                0    0
3                    1433         39.5                7                0    0
4                    1408         40.0                9                0    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

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    9661
1     339
Name: count, dtype: int64

Model Evaluation:
Accuracy: 0.985

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1932
           1       0.77      0.79      0.78        68

    accuracy                           0.98      2000
   macro avg       0.88      0.89      0.89      2000
weighted avg       0.99      0.98      0.99      2000


Confusion Matrix:
[[1916   16]
 [  14   54]]
```



```
# Install additional required packages
!pip install pyod

# Import additional libraries
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from pyod.models.knn import KNN
import seaborn as sns

## Anomaly Detection Implementation

# 1. Prepare data for anomaly detection (using the same features)
X_anomaly = df[features]

# 2. Scale the data (using same scaler as before)
X_anomaly_scaled = scaler.transform(X_anomaly)

# 3. Initialize anomaly detection models
models = {
    "Isolation Forest": IsolationForest(n_estimators=100, contamination=0.05, random_state=
    "Local Outlier Factor": LocalOutlierFactor(n_neighbors=20, contamination=0.05),
    "K-Nearest Neighbors (KNN)": KNN(contamination=0.05)
}

# 4. Fit models and detect anomalies
anomaly_results = {}
for name, model in models.items():
    if name == "Local Outlier Factor":
        anomalies = model.fit_predict(X_anomaly_scaled)
    else:
        model.fit(X_anomaly_scaled)
        anomalies = model.predict(X_anomaly_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_anomaly):.2%} of da

# 5. Add anomaly labels to dataframe
df['Isolation_Forest_Anomaly'] = anomaly_results["Isolation Forest"]
df['LOF_Anomaly'] = anomaly_results["Local Outlier Factor"]
df['KNN_Anomaly'] = anomaly_results["K-Nearest Neighbors (KNN)"]

# 6. Visualize anomalies
def plot_anomalies(feature1, feature2):
    plt.figure(figsize=(10, 6))
    sns.scatterplot(data=df, 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')
```

```
# 7. Combine anomaly detection with failure prediction
df['Combined_Risk'] = np.where(
    (df['Machine failure'] == 1) |
    (df['Isolation_Forest_Anomaly'] == 1) |
    (df['LOF_Anomaly'] == 1) |
    (df['KNN_Anomaly'] == 1),
    1, 0
)

print("\nCombined Risk (Failures + Anomalies) Distribution:")
print(df['Combined_Risk'].value_counts())

# 8. Create a risk scoring system
def calculate_risk_score(row):
    score = 0
    # Base score from failure prediction 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
    score += row['Isolation_Forest_Anomaly'] * 20
    score += row['LOF_Anomaly'] * 15
    score += row['KNN_Anomaly'] * 15

    # Cap at 100
    return min(100, score)

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

# 9. Visualize risk scores
plt.figure(figsize=(10, 6))
sns.histplot(data=df, x='Risk_Score', bins=20, kde=True)
plt.title('Distribution of Risk Scores')
plt.xlabel('Risk Score (0-100)')
plt.ylabel('Count')
plt.show()

# 10. 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())

# 11. Save the enhanced dataframe
df.to_csv('enhanced_predictive_maintenance.csv', index=False)
print("\nEnhanced dataset with anomaly detection saved to 'enhanced_predictive_maintenance.

## Create a function to evaluate new data points
def evaluate_new_data_point(data_point):
    """
    Evaluate a new data point for both failure prediction and anomalies
    Returns a comprehensive risk assessment
    """
    try:
        # Convert to dataframe
        new_data = pd.DataFrame([data_point])

        # Feature engineering
        new_data['Power'] = new_data['Torque [Nm]'] * new_data['Rotational speed [rpm]']
        new_data['Temp_diff'] = new_data['Process temperature [K]'] - new_data['Air tempera
```

```
        new_data['Torque_speed_ratio'] = new_data['Torque [Nm]'] / (new_data['Rotational sp

        # Scale features
        scaled_data = scaler.transform(new_data[features])

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

        # Get anomaly scores
        anomaly_scores = {}
        for name, model_ad in models.items():
            if name == "Local Outlier Factor":
                anomaly_scores[name] = model_ad.fit_predict(scaled_data)[0]
            else:
                anomaly_scores[name] = model_ad.predict(scaled_data)[0]

        # Calculate risk score
        risk_score = failure_prob * 50
        risk_score += sum(1 for v in anomaly_scores.values() if v == -1) * 15

        # Prepare results
        results = {
            'Failure Probability': float(failure_prob),
            'Failure Prediction': 'Likely' if failure_pred else 'Unlikely',
            'Anomaly Detection': {
                'Isolation Forest': 'Anomaly' if anomaly_scores['Isolation Forest'] == -1 e
                'Local Outlier Factor': 'Anomaly' if anomaly_scores['Local Outlier Factor']
                'KNN': 'Anomaly' if anomaly_scores['K-Nearest Neighbors (KNN)'] == 1 else '
            },
            'Overall Risk Score': min(100, risk_score),
            'Risk Level': 'High' if risk_score > 70 else ('Medium' if risk_score > 40 else
        }

        return results

    except Exception as e:
        return {'error': str(e)}

# Example usage
sample_point = df.sample(1).iloc[0]
#sample_data = X_test_scaled[0:10]

print("\nExample Evaluation:")
print(evaluate_new_data_point(sample_point))

# Save the anomaly detection models
joblib.dump(models['Isolation Forest'], 'isolation_forest.pkl')
joblib.dump(models['Local Outlier Factor'], 'local_outlier_factor.pkl')
joblib.dump(models['K-Nearest Neighbors (KNN)'], 'knn_anomaly.pkl')

print("\nAnomaly detection models saved to disk.")
```