Experiment No 8

AIM: To predict equipment failures using IoT sensor data.

Objective: Apply machine learning for predictive maintenance using Python's scikit-learn library.

Tools Used: Google Colab, Python libraries (pandas, numpy, scikit-learn, matplotlib, seaborn)

Theory:

The Industrial Internet of Things (IIoT) is revolutionizing equipment maintenance by enabling real-time monitoring of machinery using sensor data. Traditionally, maintenance approaches were either **reactive** (fixing after failure) or **preventive** (routine servicing based on schedules). However, these methods often result in unexpected failures or unnecessary maintenance costs.

Predictive maintenance, powered by machine learning, aims to predict equipment failures before they occur. It leverages historical sensor data to identify failure patterns, thereby reducing downtime, increasing operational efficiency, and lowering maintenance costs.

Machine learning (ML) is a branch of artificial intelligence (AI) that enables systems to learn from data and make predictions without being explicitly programmed. In predictive maintenance, ML models analyze vast amounts of sensor data and recognize patterns that indicate potential failures.

Key Steps in Machine Learning for Predictive Maintenance

1. Data Collection: IoT sensors collect data such as temperature, vibration, pressure, and humidity.
2. Data Preprocessing: Cleaning and transforming raw data into a suitable format for training ML models.
3. Feature Engineering: Selecting and transforming relevant sensor readings to improve model performance.
4. Model Selection & Training: Choosing an appropriate machine learning model and training it using historical data.
5. Model Evaluation: Assessing the model's accuracy using evaluation metrics.
6. Failure Prediction: Deploying the trained model to predict equipment failures in real-time.
7. Decision Support: Using predictions to schedule maintenance before failure occurs.

Machine Learning Models for Predictive Maintenance

Several machine learning models can be used to predict equipment failures. These models fall into three main categories: supervised learning, unsupervised learning, and reinforcement learning.

Supervised Learning Models

Supervised learning involves training a model on labeled data, where each sensor reading is associated with a known failure status (0 = No Failure, 1 = Failure). The model learns the relationship between input features (sensor readings) and the failure status.

Classification Models (for Predicting Failures)

Since failure prediction is a binary classification problem (failure vs. no failure), the following models are widely used:

- Logistic Regression: A simple statistical model that estimates the probability of failure based on sensor readings.
- Decision Trees: A tree-like structure where each node represents a decision based on sensor values, leading to a failure or non-failure classification.
- Random Forest: An ensemble of multiple decision trees that improves prediction accuracy by averaging results.
- Support Vector Machines (SVM): A model that finds an optimal boundary (hyperplane) between failure and non-failure instances.
- Neural Networks: Deep learning models that learn complex patterns in sensor data, useful for large datasets.

Regression Models (for Predicting Remaining Useful Life)

Instead of predicting failure as a yes/no classification, regression models estimate the remaining useful life (RUL) of a machine:

- Linear Regression: A simple model predicting RUL based on sensor values.
- Gradient Boosting (XGBoost, LightGBM): Advanced models that combine weak learners to improve predictive accuracy.

Unsupervised Learning Models

Unsupervised learning is useful when labeled failure data is not available. It detects anomalies or patterns that deviate from normal operating conditions.

- Clustering (K-Means, DBSCAN): Groups similar operating states of machines and identifies failures as outliers.
- Autoencoders (Deep Learning): Learns normal patterns and flags anomalies when deviations occur.
- Principal Component Analysis (PCA): Reduces dimensionality to highlight unusual sensor behavior.

Reinforcement Learning (RL) for Maintenance Optimization

Reinforcement Learning (RL) models learn optimal maintenance strategies by continuously interacting with an environment. They balance maintenance schedules to minimize costs while preventing failures. However, RL is less common in predictive maintenance due to high computational requirements.

Feature engineering is the process of selecting and transforming raw sensor data into meaningful inputs for machine learning models. Important features for predictive maintenance include:

- Statistical Features: Mean, standard deviation, variance of sensor readings over time.
- Time-Series Features: Rolling averages, moving windows of sensor readings.
- Frequency-Domain Features: Fourier transforms to capture vibration patterns.
- Domain-Specific Features: Sensor thresholds based on manufacturer specifications.

Proper feature selection improves model accuracy and interpretability.

Evaluation Metrics for Predictive Maintenance Models

Once a model is trained, its effectiveness is evaluated using metrics such as:

1. Accuracy: Measures the overall correctness of predictions.
2. Precision: Percentage of correctly predicted failures among all predicted failures.
3. Recall (Sensitivity): Measures how many actual failures were correctly predicted.
4. F1-Score: Balances precision and recall for better overall performance.
5. Confusion Matrix: Shows the number of true positives, false positives, true negatives, and false negatives.
6. ROC-AUC Score: Measures the model's ability to distinguish between failure and non-failure events.

A good predictive maintenance model should have high precision and recall, minimizing false positives (unnecessary maintenance) and false negatives (missed failures).

✓ Reduced Downtime: Predicts failures in advance, allowing proactive maintenance.
✓ Cost Savings: Minimizes unnecessary servicing and spare part replacements.
✓ Increased Equipment Lifespan: Detects early warning signs of wear and tear.
✓ Real-time Monitoring: Continuous analysis of sensor data ensures proactive failure prevention.
✓ Scalability: Can be applied across various industries, from manufacturing to healthcare.

Program code:

```
# Import necessary 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.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
# Set random seed for reproducibility
# np.random.seed(42)
# Generate dataset with 1000 samples
num_samples = 1000
# Simulate sensor readings
temperature = np.random.randint(30, 100, num_samples)  # Temperature in Celsius

vibration = np.round(np.random.uniform(0.1, 3.0, num_samples), 2)  # Vibration level
pressure = np.random.randint(50, 400, num_samples)  # Pressure in kPa
```

```
humidity = np.random.randint(20, 80, num_samples)  # Humidity in %
machine_age = np.random.randint(1, 20, num_samples)  # Machine age in years
# Generate failure labels (1 = Failure, 0 = No Failure) based on conditions
failure = np.where(
  (temperature > 80) & (vibration > 2.0) & (pressure > 300) & (humidity > 60),
  1,  # High failure risk
  np.random.choice([0, 1], size=num_samples, p=[0.85, 0.15])  # 15% random failures
)
# Create DataFrame
df = pd.DataFrame({
  "Temperature": temperature,
  "Vibration": vibration,
  "Pressure": pressure,
  "Humidity": humidity,
  "Machine_Age": machine_age,
  "Failure": failure
})
# Save dataset as CSV file
csv_filename = "iot_sensor_data.csv"
df.to_csv(csv_filename, index=False)

print(f"Dataset generated and saved as '{csv_filename}' successfully!")

# Display first 5 rows of the dataset
print("\nFirst 5 rows of the dataset:")
print(df.head())
# Load dataset (if running separately)
df = pd.read_csv("iot_sensor_data.csv")
# Split dataset into features and labels
X = df.drop(columns=['Failure'])  # Features
y = df['Failure']  # Target variable
# Split into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Train a Random Forest model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
# Predictions
```

```
y_pred = model.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("\nModel Accuracy:", accuracy)
print("\nClassification Report:\n", classification_report(y_test, y_pred))
# Confusion Matrix Visualization
plt.figure(figsize=(5, 4))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues',
        xticklabels=['No Failure', 'Failure'], yticklabels=['No Failure', 'Failure'])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
# Feature Importance Analysis
feature_importance = pd.Series(model.feature_importances_,
index=X.columns).sort_values(ascending=False)
plt.figure(figsize=(8, 5))
sns.barplot(x=feature_importance, y=feature_importance.index)
plt.xlabel("Feature Importance Score")
plt.ylabel("Features")
plt.title("Feature Importance in Predictive Maintenance Model")
plt.show()
```

Explanation of LOGIC:

1. Load Dataset: The IoT sensor dataset is loaded for analysis.
2. Preprocessing: Missing values are handled, and features are scaled.
3. Model Selection: Random Forest is chosen for its robustness in classification tasks.
4. Training and Testing: The dataset is split, and the model is trained using training data.
5. Prediction & Evaluation: The trained model predicts failures, and accuracy is evaluated.

Message Flow:

1. Load and preprocess IoT sensor data.
2. Train a machine learning model for failure prediction.
3. Evaluate model performance using classification metrics.
4. Use predictions for preventive maintenance planning.

Flowchart:

Start Program ↓ Load and Preprocess IoT Sensor Data ↓ Train Machine Learning Model ↓ Predict Equipment Failures ↓ Evaluate Model Performance ↓ Deploy for Predictive Maintenance ↓ End

Observation Tables:

First 5 rows of the dataset:

Sample	Temperature	Vibration	Pressure	Humidity	Machine_Age	Failure
1	46	1.82	283	24	16	0
2	63	0.12	107	39	6	0
3	86	1.34	161	74	3	0
4	66	2.72	57	46	1	0
5	42	2.76	364	31	19	0

Model Accuracy: 0.85

Classification Report

Class	Precision	Recall	F1-Score	Support (Samples)
No Failure (0)	0.85	0.99	0.92	171
Failure (1)	0.00	0.00	0.00	29
Macro Avg	0.43	0.50	0.46	200
Weighted Avg	0.73	0.85	0.79	200

Fill the table from Confusion Matrix

Actual \ Predicted	No Failure (0)	Failure (1)
No Failure (0)	170 (True Negatives)	1(False Positives)
Failure (1)	**29** (False Negatives)	**0** (True Positives)

Classification Report

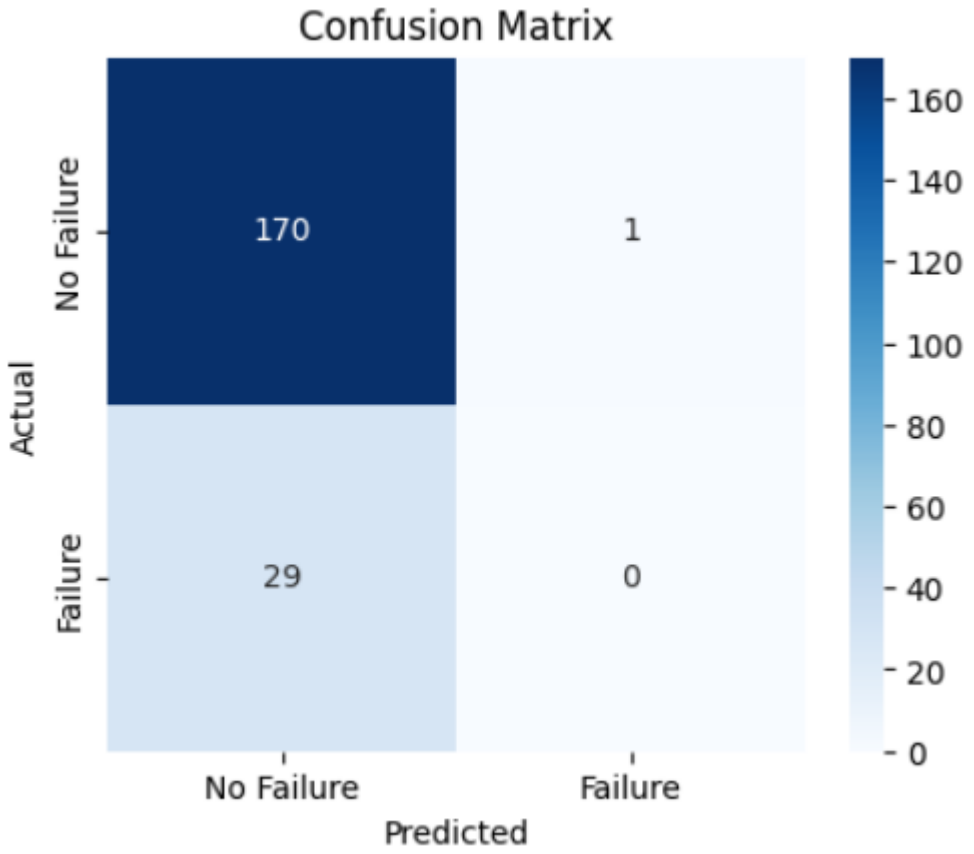
The classification report provides a detailed performance evaluation of a machine learning model. It includes precision, recall, F1-score, and support for each class (e.g., Failure vs. No Failure). Precision measures the proportion of correctly predicted positive cases out of all predicted positives, while recall (sensitivity) indicates how well the model identifies actual failures. F1-score is the harmonic mean of precision and recall, balancing both metrics. The macro average gives the unweighted mean of the scores for all classes, while the weighted average accounts for class imbalance by considering the number of actual instances per class.

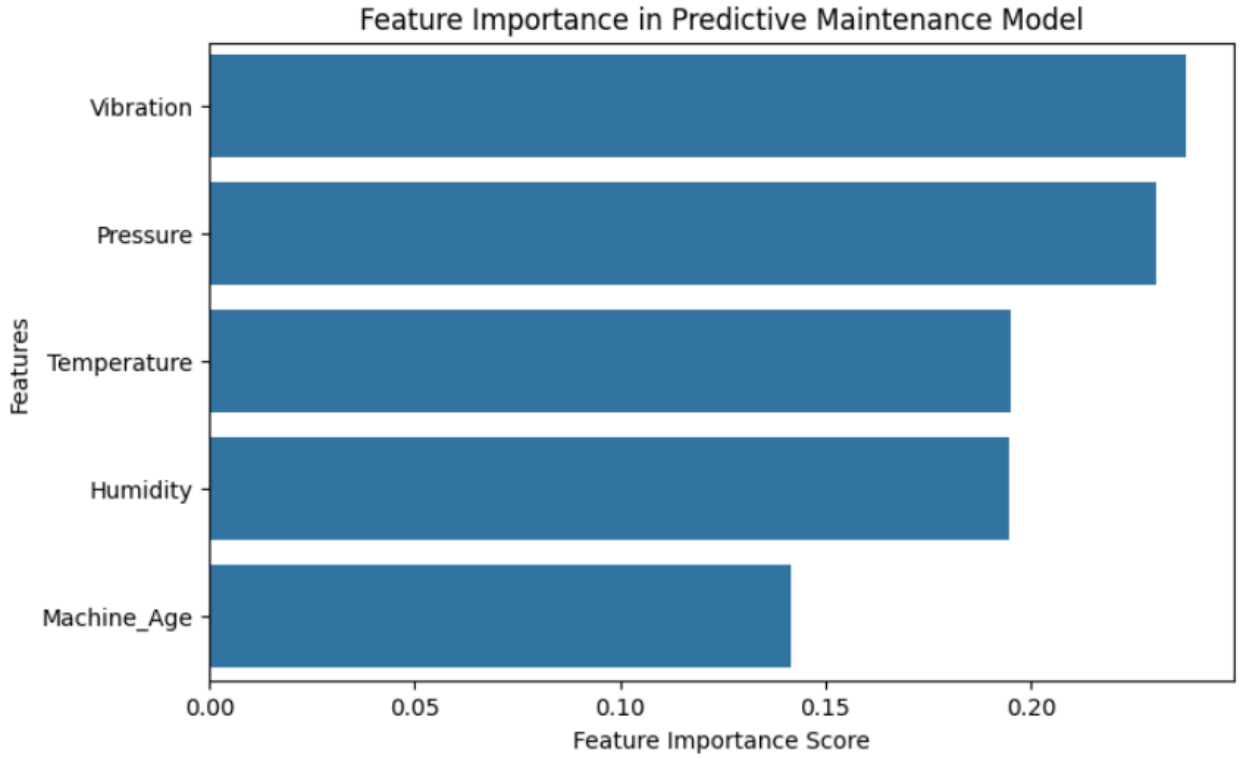
Confusion Matrix

The confusion matrix is a table that summarizes the model's predictions compared to actual labels. It consists of True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). True Positives and True Negatives represent correctly classified instances, whereas False Positives (Type I Error) indicate misclassified negatives, and False Negatives (Type II Error) represent missed failures. The confusion matrix helps visualize model accuracy and highlights areas where the model may be misclassifying, allowing for targeted improvements.

Outcome:

```
First 5 rows of the dataset:
   Temperature  Vibration  Pressure  Humidity  Machine_Age  Failure
0           46       1.82       283        24           16        0
1           63       0.12       107        39            6        0
2           86       1.34       161        74            3        0
3           66       2.72        57        46            1        0
4           42       2.76       364        31           19        0

Model Accuracy: 0.85

Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.99      0.92       171
           1       0.00      0.00      0.00        29

    accuracy                           0.85       200
   macro avg       0.43      0.50      0.46       200
weighted avg       0.73      0.85      0.79       200
```





Conclusion: Predictive maintenance using IoT sensor data and machine learning helps prevent equipment failures, reducing operational costs and downtime. The use of Random Forest provides a robust predictive model with high accuracy, making it a valuable tool for industrial applications.

Homework Assigned

AIM: To predict equipment failures using IoT sensor data with a Deep Learning model.

Objectives: Apply deep learning techniques for predictive maintenance using Python's TensorFlow/Keras library and compare its performance with the Random Forest model.

Tools Used: Google Colab, Python libraries (pandas, numpy, scikit-learn, matplotlib, seaborn, TensorFlow/Keras)

Program Code:

```
# Import necessary libraries
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Load dataset
df = pd.read_csv("iot_sensor_data.csv")

# Split dataset into features and labels
X = df.drop(columns=['Failure'])  # Features
y = df['Failure']  # Target variable

# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Build Deep Learning Model
```

```
model = Sequential([
    Dense(32, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.2),
    Dense(16, activation='relu'),
    Dropout(0.2),
    Dense(1, activation='sigmoid')
])

# Compile Model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train Model
history = model.fit(X_train, y_train, epochs=50, batch_size=16, validation_data=(X_test, y_test))

# Evaluate Model
y_pred = (model.predict(X_test) > 0.5).astype("int32")
accuracy = accuracy_score(y_test, y_pred)
print("\nModel Accuracy:", accuracy)
print("\nClassification Report:\n", classification_report(y_test, y_pred))

# Confusion Matrix Visualization
plt.figure(figsize=(5, 4))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues',
            xticklabels=['No Failure', 'Failure'], yticklabels=['No Failure', 'Failure'])
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()
```

Explanation of LOGIC:

1. Load Dataset: The IoT sensor dataset is loaded for analysis.
2. Preprocessing: Missing values are handled, and features are scaled.
3. Model Selection: Random Forest is chosen for its robustness in classification tasks.
4. Training and Testing: The dataset is split, and the model is trained using training data.
5. Prediction & Evaluation: The trained model predicts failures, and accuracy is evaluated.

Message Flow:

1. IoT sensors collect real-time data and transmit it to the data processing system.
2. Data preprocessing is performed, including cleaning, normalization, and feature selection.
3. The deep learning model processes the input features through multiple neural layers.
4. The model predicts whether an equipment failure is imminent based on the sensor inputs.
5. Predictions are evaluated using performance metrics and visualizations.

Flowchart:

Start ↓
Collect IoT Sensor Data ↓
Preprocess Data ↓
Train Neural Network ↓
Model Evaluation ↓
Predict Failures ↓
End ↓

Observation Tables:

Model Accuracy: 0.855

Classification Report

Class	Precision	Recall	F1-Score	Support (Samples)
No Failure (0)	**0.85**	**1.00**	**0.92**	**171**
Failure (1)	**0.00**	**0.00**	**0.00**	**29**
Macro Avg	**0.43**	**0.50**	**0.46**	**200**
Weighted Avg	**0.73**	**0.85**	**0.79**	**200**

Fill the table from Confusion Matrix

Actual \ Predicted	No Failure (0)	Failure (1)
No Failure (0)	171 (True Negatives)	0(False Positives)
Failure (1)	**29** (False Negatives)	**0** (True Positives)

Classification Report

The classification report provides a detailed performance evaluation of a machine learning model. It includes precision, recall, F1-score, and support for each class (e.g., Failure vs. No Failure). Precision measures the proportion of correctly predicted positive cases out of all predicted positives, while recall (sensitivity) indicates how well the model identifies actual failures. F1-score is the harmonic mean of precision and recall, balancing both metrics. The macro average gives the unweighted mean of the scores for all classes, while the weighted average accounts for class imbalance by considering the number of actual instances per class.

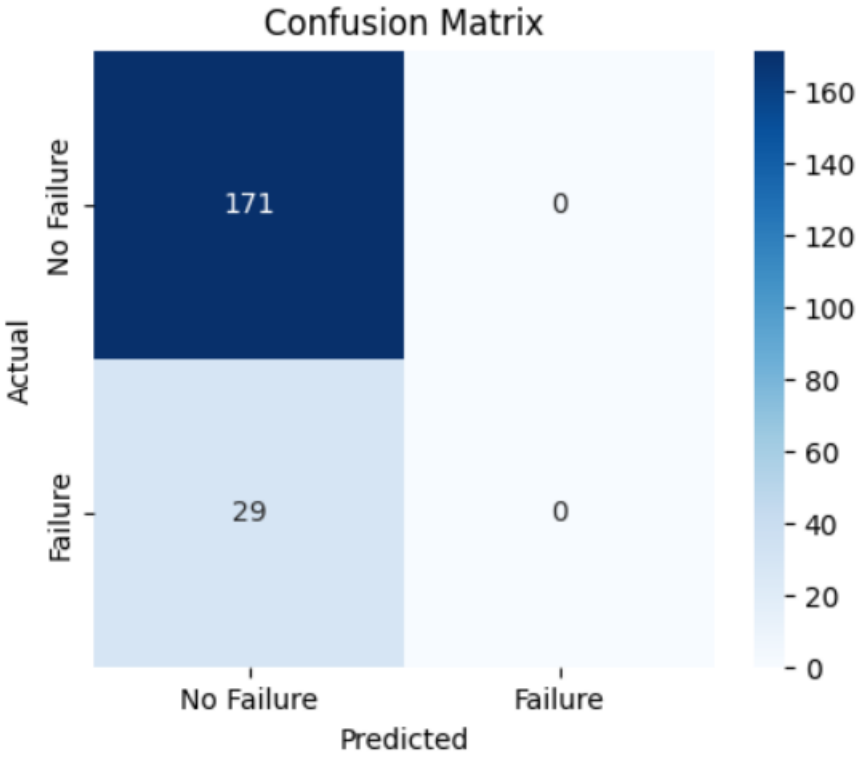
Confusion Matrix

The confusion matrix is a table that summarizes the model's predictions compared to actual labels. It consists of True Positives (TP), True Negatives (TN), False Positives (FP), and False Negatives (FN). True Positives and True Negatives represent correctly classified instances, whereas False Positives (Type I Error) indicate misclassified negatives, and False Negatives (Type II Error) represent missed failures. The confusion matrix helps visualize model accuracy and highlights areas where the model may be misclassifying, allowing for targeted improvements.

Outcome:

```
Epoch 1/50
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
50/50 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.3080 - loss: 0.7938 - val_accuracy: 0.8500 - val_loss: 0.6041
Epoch 2/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8037 - loss: 0.5875 - val_accuracy: 0.8550 - val_loss: 0.4680
Epoch 3/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8341 - loss: 0.4963 - val_accuracy: 0.8550 - val_loss: 0.4288
Epoch 4/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8341 - loss: 0.4777 - val_accuracy: 0.8550 - val_loss: 0.4245
Epoch 5/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8562 - loss: 0.4222 - val_accuracy: 0.8550 - val_loss: 0.4225
Epoch 6/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8254 - loss: 0.4872 - val_accuracy: 0.8550 - val_loss: 0.4250
Epoch 7/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8547 - loss: 0.4396 - val_accuracy: 0.8550 - val_loss: 0.4246
Epoch 8/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8526 - loss: 0.4258 - val_accuracy: 0.8550 - val_loss: 0.4230
Epoch 9/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8364 - loss: 0.4635 - val_accuracy: 0.8550 - val_loss: 0.4248
Epoch 10/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8314 - loss: 0.4647 - val_accuracy: 0.8550 - val_loss: 0.4256
Epoch 11/50
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8383 - loss: 0.4510 - val_accuracy: 0.8550 - val_loss: 0.4268
Epoch 12/50
```

```
Model Accuracy: 0.855

Classification Report:
              precision    recall  f1-score   support

           0       0.85      1.00      0.92       171
           1       0.00      0.00      0.00        29

    accuracy                           0.85       200
   macro avg       0.43      0.50      0.46       200
weighted avg       0.73      0.85      0.79       200
```



Conclusion: By implementing a neural network model using TensorFlow/Keras, predictive maintenance accuracy can be further optimized. Deep learning enables better failure prediction by capturing intricate sensor data patterns, thus improving equipment reliability and reducing costs.