Connected to unified-ai-pipeline_v2 (Python 3.9.18)

In [ ]:
```
import pandas as pd
```

In [ ]:
```
import numpy as np
```

In [ ]:
```
import os
```

In [ ]:
```
import matplotlib.pyplot as plt
```

In [ ]:
```
import seaborn as sns
```

In [ ]:
```
import logging
```

In [ ]:
```
def load_sensor_data(file_path):
    """Loads a processed CSV file."""
    try:
        df = pd.read_csv(file_path)
        print(f"✅ Successfully loaded {file_path}")
        return df
    except Exception as e:
        print(f"❌ Error loading {file_path}: {e}")
        return None
```

In [ ]:
```
saw_csv_path = r"C:\DT_Projects\Unified-AI-Pipeline-for-Predictive-Maintenance-a
```

In [ ]:
```
lathe_csv_path = r"C:\DT_Projects\Unified-AI-Pipeline-for-Predictive-Maintenance
```

In [ ]:
```
saw_df = load_sensor_data(saw_csv_path)
```

```
✅ Successfully loaded C:\DT_Projects\Unified-AI-Pipeline-for-Predictive-Mainten
ance-and-Optimization\data\processed\CYLINDER_BOTTOM_SAW\saw_internal_signal.csv
```

In [ ]:
```
lathe_df = load_sensor_data(lathe_csv_path)
```

```
✅ Successfully loaded C:\DT_Projects\Unified-AI-Pipeline-for-Predictive-Mainten
ance-and-Optimization\data\processed\PISTON_ROD_CNC_LATHE\cnc_lathe_internal_sign
al.csv
```

In [ ]:
```
saw_df["timestamp"] = pd.to_datetime(saw_df["timestamp"])
```

In [ ]:
```
lathe_df["timestamp"] = pd.to_datetime(lathe_df["timestamp"])
```

In [ ]:
```
print(saw_df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 332847 entries, 0 to 332846
Data columns (total 46 columns):
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   timestamp                    332847 non-null  datetime64[ns]
 1   CPU_cooler_temp              332847 non-null  float64
 2   CPU_temp                     332847 non-null  float64
 3   CutCounter                   332847 non-null  float64
 4   CutTime                      332847 non-null  float64
 5   FFT_requirement              332847 non-null  float64
 6   FlatstreamCutCounter         332847 non-null  float64
 7   FlatstreamDone               332847 non-null  float64
 8   FsMode_1Raw_2FftRaw_3FttHK   332847 non-null  float64
 9   lift_active                  332847 non-null  float64
 10  motor_on                     332847 non-null  float64
 11  PData.CosPhi                 332847 non-null  float64
 12  PData.CutEnergy              332847 non-null  float64
 13  PData.PEff                   332847 non-null  float64
 14  P_feed                       332847 non-null  float64
 15  Position                     332847 non-null  float64
 16  blade_position               332847 non-null  float64
 17  TData.T1                     332847 non-null  float64
 18  TData.T2                     332847 non-null  float64
 19  TData.T3                     332847 non-null  float64
 20  TData.T4                     332847 non-null  float64
 21  TData.T_IR                   332847 non-null  float64
 22  Vib01.CREST                  332847 non-null  float64
 23  Vib01.Kurtosis               332847 non-null  float64
 24  Vib01.Peak                   332847 non-null  float64
 25  Vib01.RMS                    332847 non-null  float64
 26  Vib01.Skewness               332847 non-null  float64
 27  Vib01.VDI3832                332847 non-null  float64
 28  Vib02.CREST                  332847 non-null  float64
 29  Vib02.Kurtosis               332847 non-null  float64
 30  Vib02.Peak                   332847 non-null  float64
 31  Vib02.RMS                    332847 non-null  float64
 32  Vib02.Skewness               332847 non-null  float64
 33  Vib02.VDI3832                332847 non-null  float64
 34  Vib03.CREST                  332847 non-null  float64
 35  Vib03.Kurtosis               332847 non-null  float64
 36  Vib03.Peak                   332847 non-null  float64
 37  Vib03.RMS                    332847 non-null  float64
 38  Vib03.Skewness               332847 non-null  float64
 39  Vib03.VDI3832                332847 non-null  float64
 40  teeth_per_blade              332847 non-null  float64
 41  bCutActive                   332847 non-null  float64
 42  f_light_barrier              332847 non-null  float64
 43  top_material_edge            332847 non-null  float64
 44  v_feed                       332847 non-null  float64
 45  anomaly                      332847 non-null  int64
dtypes: datetime64[ns](1), float64(44), int64(1)
memory usage: 116.8 MB
None
```

In [ ]:
```
print(saw_df.shape, lathe_df.shape)
```

```
(332847, 46) (504826, 92)
```

In [ ]:
```python
def handle_missing_values(df):
    """Fills missing values & drops columns with >50% missing values."""
    threshold = len(df) * 0.5
    df = df.dropna(thresh=threshold, axis=1)

    for column in df.columns:
        if df[column].dtype == "object":
            df[column] = df[column].fillna(method="ffill")
        else:
            df[column] = df[column].fillna(df[column].median())

    return df
```

In [ ]:
```python
saw_df = handle_missing_values(saw_df)
```

In [ ]:
```python
lathe_df = handle_missing_values(lathe_df)
```

In [ ]:
```python
def remove_correlated_features(df, correlation_threshold=0.9):
    """Removes features that are highly correlated."""
    numeric_df = df.select_dtypes(include=[np.number])
    corr_matrix = numeric_df.corr().abs()
    upper_triangle = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).
    to_drop = [column for column in upper_triangle.columns if any(upper_triangle

    df = df.drop(columns=to_drop)
    return df
```

In [ ]:
```python
saw_df = remove_correlated_features(saw_df)
```

In [ ]:
```python
lathe_df = remove_correlated_features(lathe_df)
```

In [ ]:
```python
def scale_features(df):
    """Applies standard scaling to numerical features."""
    scaler = StandardScaler()
    numeric_cols = df.select_dtypes(include=[np.number]).columns
    df[numeric_cols] = scaler.fit_transform(df[numeric_cols])
    logging.info(f"✅ Data scaled successfully. {len(numeric_cols)} features st
    return df, scaler
```

In [ ]:
```python
def handle_missing_values(df):
    """Handles missing values by dropping columns with >50% missing data and fil
    threshold = len(df) * 0.5
    df = df.dropna(thresh=threshold, axis=1)

    for column in df.columns:
        if df[column].dtype == "object":
            df[column] = df[column].fillna(method="ffill")
        else:
            df[column] = df[column].fillna(df[column].median())

    logging.info("✅ Missing values handled successfully.")
    return df
```

In [ ]:
```python
cylinder_bottom_saw_df = handle_missing_values(saw_df)
```

In [ ]:
```python
piston_rod_cnc_lathe_df = handle_missing_values(lathe_df)
```

In [ ]:
```
cylinder_bottom_saw_df = remove_correlated_features(cylinder_bottom_saw_df)
```

In [ ]:
```
piston_rod_cnc_lathe_df = remove_correlated_features(piston_rod_cnc_lathe_df)
```

In [ ]:
```
cylinder_bottom_saw_df.head()
```

Out[ ]:

	timestamp	CPU_cooler_temp	CutCounter	FFT_requirement	FlatstreamDone	FsMod
0	2022-08-16 05:59:47.003	636.0	5527.0	0.0	1.0	
1	2022-08-16 05:59:47.503	636.0	5527.0	0.0	1.0	
2	2022-08-16 05:59:48.003	636.0	5527.0	0.0	1.0	
3	2022-08-16 05:59:48.503	636.0	5527.0	0.0	1.0	
4	2022-08-16 05:59:49.005	636.0	5527.0	0.0	1.0	

5 rows × 37 columns

In [ ]:
```
piston_rod_cnc_lathe_df.head()
```

Out[ ]:

	timestamp	NCLine	ProgramName	ProgramStatus	SpindleRPM	TimeSinceSt
0	2022-09-06 09:33:24.108042002	100.0	0.0	2.0	0.0	
1	2022-09-06 09:33:24.257301092	100.0	0.0	2.0	0.0	
2	2022-09-06 09:33:24.509967089	100.0	0.0	2.0	0.0	
3	2022-09-06 09:33:24.654723883	100.0	0.0	2.0	0.0	
4	2022-09-06 09:33:24.911717892	100.0	0.0	2.0	0.0	

5 rows × 62 columns

In [ ]:
```
def scale_features(df):
    """Applies standard scaling to numerical features."""
    scaler = StandardScaler()
    numeric_cols = df.select_dtypes(include=[np.number]).columns  # Select only
    df[numeric_cols] = scaler.fit_transform(df[numeric_cols])  # Apply scaling
    logging.info(f"✅ Data scaled successfully. {len(numeric_cols)} features st
    return df, scaler
```

In [ ]:
```
def scale_features(df):
    """Applies standard scaling to numerical features."""
```

```
    scaler = StandardScaler()   # Initialize StandardScaler
    numeric_cols = df.select_dtypes(include=[np.number]).columns  # Select only
    df[numeric_cols] = scaler.fit_transform(df[numeric_cols])  # Apply scaling
    logging.info(f"✅ Data scaled successfully. {len(numeric_cols)} features st
    return df, scaler  # Return
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler
```

In [ ]:
```
cylinder_bottom_saw_df, saw_scaler = scale_features(cylinder_bottom_saw_df)
```

In [ ]:
```
piston_rod_cnc_lathe_df, lathe_scaler = scale_features(piston_rod_cnc_lathe_df)
```

In [ ]:
```
joblib.dump(saw_scaler, "models/saw_scaler.pkl")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
File c:\DT_Projects\Unified-AI-Pipeline-for-Predictive-Maintenance-and-Optimizati
on\src\models\train_model_v3.py:1
----> 1 joblib.dump(saw_scaler, "models/saw_scaler.pkl")

NameError: name 'joblib' is not defined
```

In [ ]:
```
import os  # File paths
```

In [ ]:
```
import sys  # System-specific parameters
```

In [ ]:
```
import logging # to log messages
```

In [ ]:
```
import pandas as pd
```

In [ ]:
```
import numpy as np
```

In [ ]:
```
import matplotlib.pyplot as plt
```

In [ ]:
```
import seaborn as sns
```

In [ ]:
```
import joblib   # to save and load models
```

In [ ]:
```
import tensorflow as tf
```

In [ ]:
```
from tensorflow.keras.models import load_model
```

In [ ]:
```
from sklearn.decomposition import PCA
```

In [ ]:
```
from sklearn.preprocessing import StandardScaler
```

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [ ]:
```
from sklearn.svm import OneClassSVM
```

In [ ]:
```
from sklearn.metrics import (
    classification_report, confusion_matrix, accuracy_score, roc_curve, auc)
```

In [ ]:
```
from mpl_toolkits.mplot3d import Axes3D
```

In [ ]:
```
import plotly.express as px
```

In [ ]:
```
joblib.dump(saw_scaler, "saw_scaler.pkl.joblib")
```

Out[ ]:
```
['saw_scaler.pkl.joblib']
```

In [ ]:
```
joblib.dump(lathe_scaler, "lathe_scaler.pkl.joblib")
```

Out[ ]:
```
['lathe_scaler.pkl.joblib']
```

In [ ]:
```
print(cylinder_bottom_saw_df[['timestamp', 'anomaly']].head())
```

```
                timestamp   anomaly
0 2022-08-16 05:59:47.003 -0.226364
1 2022-08-16 05:59:47.503 -0.226364
2 2022-08-16 05:59:48.003 -0.226364
3 2022-08-16 05:59:48.503 -0.226364
4 2022-08-16 05:59:49.005 -0.226364
```

In [ ]:
```
print(piston_rod_cnc_lathe_df[['timestamp', 'anomaly']].head())
```

```
                    timestamp   anomaly
0 2022-09-06 09:33:24.108042002 -0.341449
1 2022-09-06 09:33:24.257301092 -0.341449
2 2022-09-06 09:33:24.509967089 -0.341449
3 2022-09-06 09:33:24.654723883 -0.341449
4 2022-09-06 09:33:24.911717892 -0.341449
```

In [ ]:
```
def scale_features(df):
    """Applies standard scaling to numerical features."""
    scaler = StandardScaler()   # Initialize StandardScaler
    numeric_cols = df.select_dtypes(include=[np.number]).columns  # Select only
    df[numeric_cols] = scaler.fit_transform(df[numeric_cols])  # Apply scaling
    logging.info(f"✅ Data scaled successfully. {len(numeric_cols)} features st
    return df, scaler  # Return the transformed dataframe and the scaler used
```

In [ ]:
```
cylinder_bottom_saw_df, saw_scaler = scale_features(cylinder_bottom_saw_df)
```

In [ ]:
```
piston_rod_cnc_lathe_df, lathe_scaler = scale_features(piston_rod_cnc_lathe_df)
```

In [ ]:
```
print(cylinder_bottom_saw_df[['timestamp', 'anomaly']].head())
```

```
                timestamp   anomaly
0 2022-08-16 05:59:47.003 -0.226364
1 2022-08-16 05:59:47.503 -0.226364
2 2022-08-16 05:59:48.003 -0.226364
3 2022-08-16 05:59:48.503 -0.226364
4 2022-08-16 05:59:49.005 -0.226364
```

In [ ]:
```
print(piston_rod_cnc_lathe_df[['timestamp', 'anomaly']].head())
```

```
                    timestamp   anomaly
0 2022-09-06 09:33:24.108042002 -0.341449
1 2022-09-06 09:33:24.257301092 -0.341449
2 2022-09-06 09:33:24.509967089 -0.341449
3 2022-09-06 09:33:24.654723883 -0.341449
4 2022-09-06 09:33:24.911717892 -0.341449
```

In [ ]:
```
def scale_features(df):
    """Applies standard scaling to numerical features, excluding 'anomaly' and '
    scaler = StandardScaler()

    # Exclude 'timestamp' and 'anomaly' from scaling
    exclude_cols = ['timestamp', 'anomaly']
    numeric_cols = [col for col in df.select_dtypes(include=[np.number]).columns

    # Apply scaling only to selected columns
    df[numeric_cols] = scaler.fit_transform(df[numeric_cols])

    logging.info(f"✅ Data scaled successfully. {len(numeric_cols)} features st
    return df, scaler
```

In [ ]:
```
cylinder_bottom_saw_df, saw_scaler = scale_features(cylinder_bottom_saw_df)
```

In [ ]:
```
piston_rod_cnc_lathe_df, lathe_scaler = scale_features(piston_rod_cnc_lathe_df)
```

In [ ]:
```
print(cylinder_bottom_saw_df.head())
```

```
                timestamp  CPU_cooler_temp  CutCounter  FFT_requirement  \
0 2022-08-16 05:59:47.003         1.263835   -2.049671              0.0
1 2022-08-16 05:59:47.503         1.263835   -2.049671              0.0
2 2022-08-16 05:59:48.003         1.263835   -2.049671              0.0
3 2022-08-16 05:59:48.503         1.263835   -2.049671              0.0
4 2022-08-16 05:59:49.005         1.263835   -2.049671              0.0

   FlatstreamDone  FsMode_1Raw_2FftRaw_3FttHK  lift_active  motor_on  \
0        0.591068                         0.0    -0.202611  0.063244
1        0.591068                         0.0    -0.202611  0.063244
2        0.591068                         0.0    -0.202611  0.063244
3        0.591068                         0.0    -0.202611  0.063244
4        0.591068                         0.0    -0.202611  0.063244

   PData.CosPhi    P_feed  ...  Vib03.Peak  Vib03.RMS  Vib03.Skewness  \
0     -1.993444 -0.190140  ...   -0.515804   -1.13463        0.032657
1     -2.008497 -0.211096  ...   -0.516255   -1.13463        0.050856
2     -1.993444 -0.214876  ...   -0.536937   -1.13463        0.145779
3     -1.993444 -0.207661  ...   -0.547697   -1.13463        0.181659
4     -1.993444 -0.204226  ...   -0.547825   -1.13463        0.197170

   Vib03.VDI3832  teeth_per_blade  bCutActive  f_light_barrier  \
0            0.0              0.0   -1.380633         2.817201
1            0.0              0.0   -1.380633         2.768843
2            0.0              0.0   -1.380633         2.768843
3            0.0              0.0   -1.380633         2.817201
4            0.0              0.0   -1.380633         2.817201

   top_material_edge    v_feed   anomaly
0                0.0  0.044715 -0.226364
1                0.0  0.044715 -0.226364
2                0.0  0.116280 -0.226364
3                0.0  0.116280 -0.226364
4                0.0  0.116280 -0.226364

[5 rows x 37 columns]
```

In [ ]:
```
print(piston_rod_cnc_lathe_df.head())
```

```
                      timestamp    NCLine  ProgramName  ProgramStatus  \
0 2022-09-06 09:33:24.108042002 -1.943439          0.0      -2.753656
1 2022-09-06 09:33:24.257301092 -1.943439          0.0      -2.753656
2 2022-09-06 09:33:24.509967089 -1.943439          0.0      -2.753656
3 2022-09-06 09:33:24.654723883 -1.943439          0.0      -2.753656
4 2022-09-06 09:33:24.911717892 -1.943439          0.0      -2.753656

   SpindleRPM  TimeSinceStartup   aaCurr1  aaCurr13  aaCurr14  aaCurr16  ...  \
0    0.005127         -1.238662  0.020861  0.270917  0.479454 -0.154296  ...
1    0.005127         -1.238662  0.020861  0.268659  0.480829 -0.163583  ...
2    0.005127         -1.238662  0.020170  0.267567  0.479454 -0.122241  ...
3    0.005127         -1.238662  0.021559  0.265382  0.480142 -0.090186  ...
4    0.005127         -1.238662  0.022257  0.268659  0.481516 -0.103966  ...

   measPos113  measPos114  measPos116  measPos12  measPos16  measPos17  \
0    1.405264    0.625511    0.289637  -1.390763   1.973848   0.787579
1    1.405264    0.625511    0.289647  -1.390763   1.973848   0.787579
2    1.405264    0.625511    0.289647  -1.390763   1.973848   0.787579
3    1.405264    0.625511    0.289637  -1.390763   1.973848   0.787579
4    1.405264    0.625511    0.289637  -1.390763   1.973848   0.787579

   measPos18  measPos19  unknown_91   anomaly
0   4.034325  -1.792383         0.0 -0.341449
1   4.034325  -1.792383         0.0 -0.341449
2   4.034325  -1.792383         0.0 -0.341449
3   4.034325  -1.792383         0.0 -0.341449
4   4.034325  -1.792383         0.0 -0.341449

[5 rows x 62 columns]
```

In [ ]:
```
print(cylinder_bottom_saw_df[['timestamp', 'anomaly']].head())
```

```
                timestamp   anomaly
0 2022-08-16 05:59:47.003 -0.226364
1 2022-08-16 05:59:47.503 -0.226364
2 2022-08-16 05:59:48.003 -0.226364
3 2022-08-16 05:59:48.503 -0.226364
4 2022-08-16 05:59:49.005 -0.226364
```

In [ ]:
```
print(piston_rod_cnc_lathe_df[['timestamp', 'anomaly']].head())
```

```
                      timestamp   anomaly
0 2022-09-06 09:33:24.108042002 -0.341449
1 2022-09-06 09:33:24.257301092 -0.341449
2 2022-09-06 09:33:24.509967089 -0.341449
3 2022-09-06 09:33:24.654723883 -0.341449
4 2022-09-06 09:33:24.911717892 -0.341449
```

In [ ]:
```
def scale_features(df):
    """Applies standard scaling to numerical features, excluding 'anomaly' and '
    scaler = StandardScaler()

    # Exclude 'timestamp' and 'anomaly' from scaling
    exclude_cols = ['timestamp', 'anomaly']
    numeric_cols = [col for col in df.select_dtypes(include=[np.number]).columns

    # Apply scaling only to selected columns
    df[numeric_cols] = scaler.fit_transform(df[numeric_cols])

    logging.info(f"✅ Data scaled successfully. {len(numeric_cols)} features st
    return df, scaler
```

In [ ]:
```
cylinder_bottom_saw_df, saw_scaler = scale_features(cylinder_bottom_saw_df)
```

In [ ]:
```
piston_rod_cnc_lathe_df, lathe_scaler = scale_features(piston_rod_cnc_lathe_df)
```

In [ ]:
```
print(cylinder_bottom_saw_df[['timestamp', 'anomaly']].head())
```

```
                timestamp   anomaly
0 2022-08-16 05:59:47.003 -0.226364
1 2022-08-16 05:59:47.503 -0.226364
2 2022-08-16 05:59:48.003 -0.226364
3 2022-08-16 05:59:48.503 -0.226364
4 2022-08-16 05:59:49.005 -0.226364
```

In [ ]:
```
print(piston_rod_cnc_lathe_df[['timestamp', 'anomaly']].head())
```

```
                    timestamp   anomaly
0 2022-09-06 09:33:24.108042002 -0.341449
1 2022-09-06 09:33:24.257301092 -0.341449
2 2022-09-06 09:33:24.509967089 -0.341449
3 2022-09-06 09:33:24.654723883 -0.341449
4 2022-09-06 09:33:24.911717892 -0.341449
```

In [ ]:
```
def scale_features(df):
    """Applies standard scaling to numerical features, explicitly preserving 'ti

    scaler = StandardScaler()

    # Make a copy to avoid modifying the original dataframe
    df_scaled = df.copy()

    # Exclude 'timestamp' and 'anomaly' from scaling
    exclude_cols = ['timestamp', 'anomaly']
    numeric_cols = [col for col in df.select_dtypes(include=[np.number]).columns

    # Apply StandardScaler only to the selected numeric columns
    df_scaled[numeric_cols] = scaler.fit_transform(df_scaled[numeric_cols])

    # Ensure 'anomaly' column is preserved correctly
    df_scaled['anomaly'] = df['anomaly'].astype(int)  # Convert back to integer

    logging.info(f"✅ Data scaled successfully. {len(numeric_cols)} features st
    return df_scaled, scaler
```

In [ ]:
```
cylinder_bottom_saw_df, saw_scaler = scale_features(cylinder_bottom_saw_df)
```

In [ ]:
```
piston_rod_cnc_lathe_df, lathe_scaler = scale_features(piston_rod_cnc_lathe_df)
```

In [ ]:
```
print(cylinder_bottom_saw_df[['timestamp', 'anomaly']].head())
```

```
                timestamp  anomaly
0 2022-08-16 05:59:47.003        0
1 2022-08-16 05:59:47.503        0
2 2022-08-16 05:59:48.003        0
3 2022-08-16 05:59:48.503        0
4 2022-08-16 05:59:49.005        0
```

In [ ]:
```
print(piston_rod_cnc_lathe_df[['timestamp', 'anomaly']].head())
```

```
                       timestamp  anomaly
0 2022-09-06 09:33:24.108042002        0
1 2022-09-06 09:33:24.257301092        0
2 2022-09-06 09:33:24.509967089        0
3 2022-09-06 09:33:24.654723883        0
4 2022-09-06 09:33:24.911717892        0
```

In [ ]:
```
print(cylinder_bottom_saw_df[['timestamp', 'anomaly']].tail())
```

```
                        timestamp  anomaly
332842 2022-10-31 14:06:22.503000        4
332843 2022-10-31 14:06:23.003000        4
332844 2022-10-31 14:06:23.503999        4
332845 2022-10-31 14:06:24.002000        4
332846 2022-10-31 14:06:24.503000        4
```

In [ ]:
```
print(piston_rod_cnc_lathe_df[['timestamp', 'anomaly']].tail())
```

```
                           timestamp  anomaly
504821 2022-11-04 15:13:42.100191116        0
504822 2022-11-04 15:13:42.267194033        0
504823 2022-11-04 15:13:42.411525965        0
504824 2022-11-04 15:13:42.670864105        0
504825 2022-11-04 15:13:42.830426931        0
```

In [ ]:
```
def scale_features(df):
    """Applies standard scaling to numerical features, explicitly preserving 'ti

    scaler = StandardScaler()

    # Make a copy to avoid modifying the original dataframe
    df_scaled = df.copy()

    # Exclude 'timestamp' and 'anomaly' from scaling
    exclude_cols = ['timestamp', 'anomaly']
    numeric_cols = [col for col in df.select_dtypes(include=[np.number]).columns

    # Apply StandardScaler only to the selected numeric columns
    df_scaled[numeric_cols] = scaler.fit_transform(df_scaled[numeric_cols])

    # Ensure 'anomaly' column is strictly binary (0 or 1)
    df_scaled['anomaly'] = df['anomaly'].apply(lambda x: 1 if x > 0 else 0)

    logging.info(f"✅ Data scaled successfully. {len(numeric_cols)} features st
    return df_scaled, scaler
```

In [ ]:
```
cylinder_bottom_saw_df, saw_scaler = scale_features(cylinder_bottom_saw_df)
```

In [ ]:
```
piston_rod_cnc_lathe_df, lathe_scaler = scale_features(piston_rod_cnc_lathe_df)
```

In [ ]:
```
print(cylinder_bottom_saw_df[['timestamp', 'anomaly']].tail())
```

```
                       timestamp  anomaly
332842 2022-10-31 14:06:22.503000        1
332843 2022-10-31 14:06:23.003000        1
332844 2022-10-31 14:06:23.503999        1
332845 2022-10-31 14:06:24.002000        1
332846 2022-10-31 14:06:24.503000        1
```

In [ ]:
```
print(piston_rod_cnc_lathe_df[['timestamp', 'anomaly']].tail())
```

```
                          timestamp  anomaly
504821 2022-11-04 15:13:42.100191116        0
504822 2022-11-04 15:13:42.267194033        0
504823 2022-11-04 15:13:42.411525965        0
504824 2022-11-04 15:13:42.670864105        0
504825 2022-11-04 15:13:42.830426931        0
```

In [ ]:
```
print(cylinder_bottom_saw_df[['timestamp', 'anomaly']].head())
```

```
                timestamp  anomaly
0 2022-08-16 05:59:47.003        0
1 2022-08-16 05:59:47.503        0
2 2022-08-16 05:59:48.003        0
3 2022-08-16 05:59:48.503        0
4 2022-08-16 05:59:49.005        0
```

In [ ]:
```
print(piston_rod_cnc_lathe_df[['timestamp', 'anomaly']].head())
```

```
                     timestamp  anomaly
0 2022-09-06 09:33:24.108042002        0
1 2022-09-06 09:33:24.257301092        0
2 2022-09-06 09:33:24.509967089        0
3 2022-09-06 09:33:24.654723883        0
4 2022-09-06 09:33:24.911717892        0
```

In [ ]:
```
print(cylinder_bottom_saw_df[['timestamp', 'anomaly']].tail())
```

```
                       timestamp  anomaly
332842 2022-10-31 14:06:22.503000        1
332843 2022-10-31 14:06:23.003000        1
332844 2022-10-31 14:06:23.503999        1
332845 2022-10-31 14:06:24.002000        1
332846 2022-10-31 14:06:24.503000        1
```

In [ ]:
```
print(piston_rod_cnc_lathe_df[['timestamp', 'anomaly']].tail())
```

```
                          timestamp  anomaly
504821 2022-11-04 15:13:42.100191116        0
504822 2022-11-04 15:13:42.267194033        0
504823 2022-11-04 15:13:42.411525965        0
504824 2022-11-04 15:13:42.670864105        0
504825 2022-11-04 15:13:42.830426931        0
```

In [ ]:
```
saw_anomaly_count = cylinder_bottom_saw_df['anomaly'].value_counts()
```

In [ ]:
```
lathe_anomaly_count = piston_rod_cnc_lathe_df['anomaly'].value_counts()
```

In [ ]:
```
anomaly_counts = pd.DataFrame({
    "Machine": ["Cylinder Bottom Saw", "CNC Lathe (Piston Rod)"],
    "Normal Count (0)": [saw_anomaly_count.get(0, 0), lathe_anomaly_count.get(0,
    "Anomaly Count (1)": [saw_anomaly_count.get(1, 0), lathe_anomaly_count.get(1
})
```

In [ ]:
```
print(anomaly_counts)
```

```
                 Machine  Normal Count (0)  Anomaly Count (1)
0     Cylinder Bottom Saw            316623              16224
1  CNC Lathe (Piston Rod)            452115              52711
```

In [ ]:
```
joblib.dump(saw_scaler, "saw_scaler.pkl.joblib")
```

Out[ ]:
```
['saw_scaler.pkl.joblib']
```

In [ ]:
```
joblib.dump(lathe_scaler, "lathe_scaler.pkl.joblib")
```

Out[ ]:
```
['lathe_scaler.pkl.joblib']
```

In [ ]:
```python
def apply_pca(df, variance_threshold=0.95):
    """
    Applies PCA on the normal data and transforms the entire dataset.

    Args:
        df (pd.DataFrame): The dataset containing all instances (normal + anomal
        variance_threshold (float): The amount of variance to retain in PCA.

    Returns:
        tuple: (PCA object, PCA-transformed DataFrame)
    """
    # Separate normal data (anomaly = 0)
    normal_data = df[df["anomaly"] == 0].drop(columns=["anomaly", "timestamp"],

    # Standardize the features before applying PCA
    scaler = StandardScaler()
    normal_data_scaled = scaler.fit_transform(normal_data)

    # Apply PCA
    pca = PCA(n_components=variance_threshold)
    normal_pca_transformed = pca.fit_transform(normal_data_scaled)

    # Apply trained PCA transformation to the full dataset (including anomalies)
    full_data_scaled = scaler.transform(df.drop(columns=["anomaly", "timestamp"]
    full_pca_transformed = pca.transform(full_data_scaled)

    # Convert transformed data into a DataFrame
    pca_columns = [f"PC{i+1}" for i in range(full_pca_transformed.shape[1])]
    pca_df = pd.DataFrame(full_pca_transformed, columns=pca_columns)

    # Add back anomaly labels and timestamps
    pca_df["timestamp"] = df["timestamp"].values
    pca_df["anomaly"] = df["anomaly"].values

    # Print explained variance
    explained_variance = sum(pca.explained_variance_ratio_)
    logging.info(f"✅ PCA applied. Retained Variance: {explained_variance:.4f}"

    return pca, pca_df, scaler
```

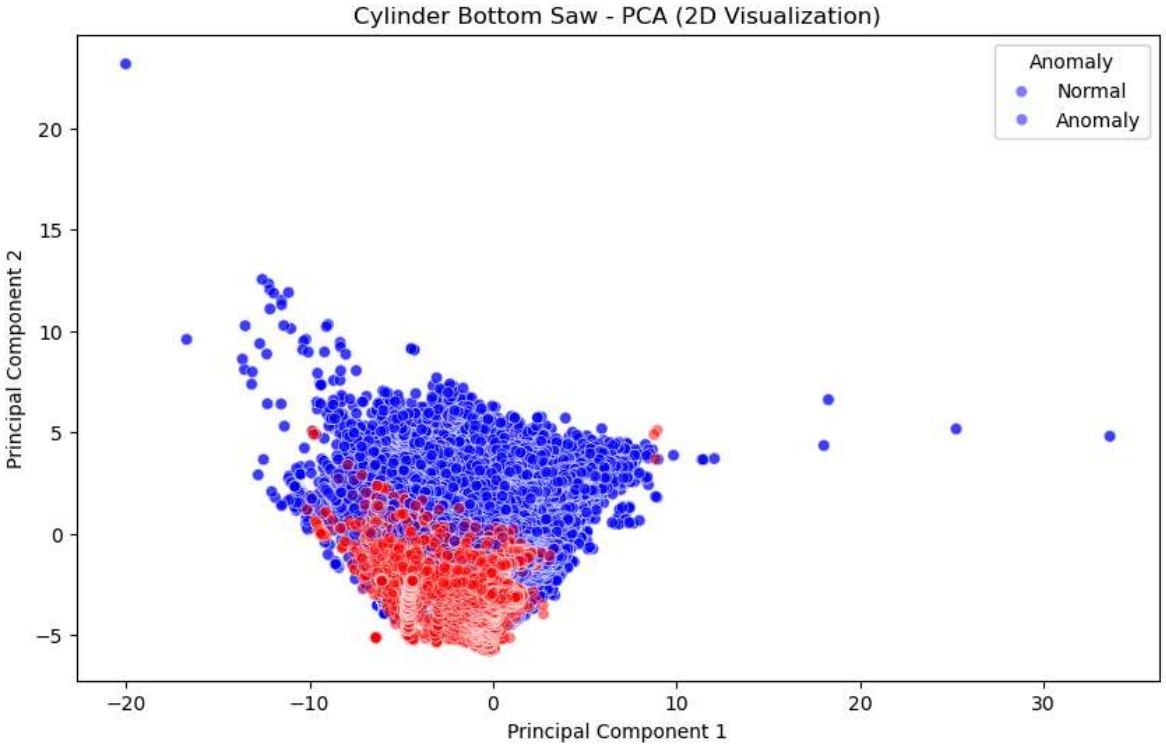
In [ ]:
```python
pca_saw, saw_pca_df, saw_scaler = apply_pca(cylinder_bottom_saw_df)
```

In [ ]:
```python
pca_lathe, lathe_pca_df, lathe_scaler = apply_pca(piston_rod_cnc_lathe_df)
```

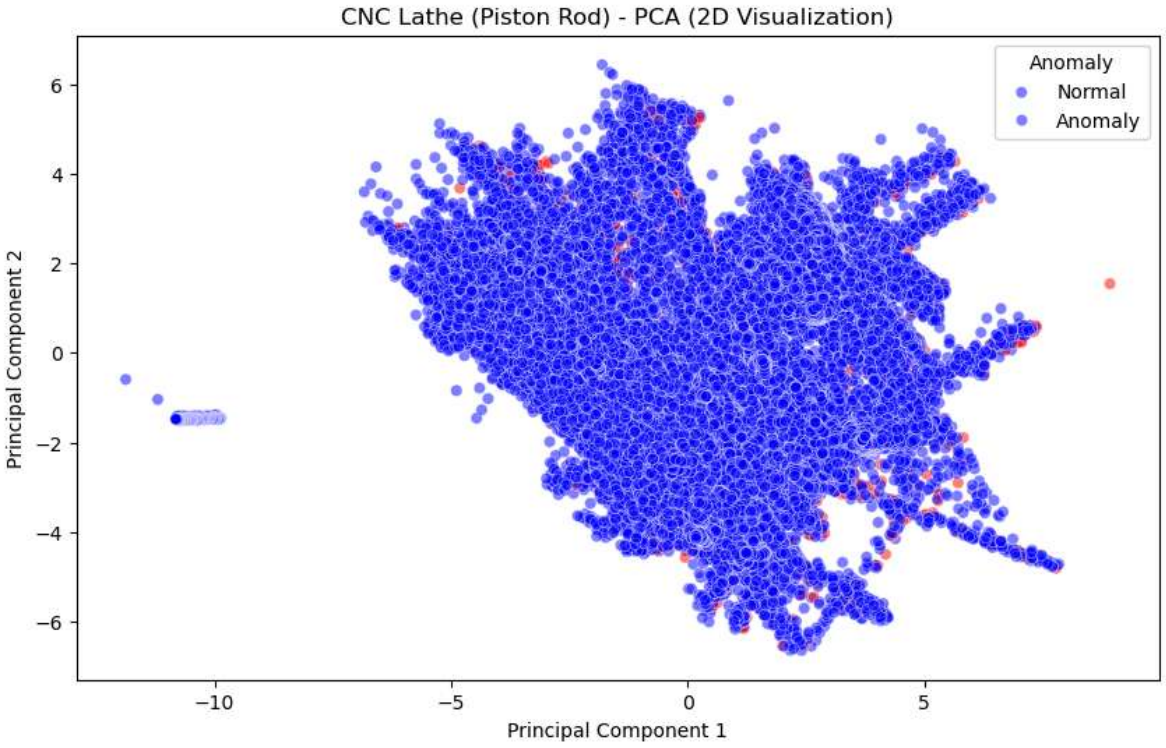
In [ ]:
```python
def plot_pca_2d(pca_df, title):
    """
    Plots a 2D scatter plot of the first two principal components.

    Args:
        pca_df (pd.DataFrame): PCA-transformed DataFrame containing "PC1", "PC2"
        title (str): Title of the plot.
    """
    plt.figure(figsize=(10, 6))
    sns.scatterplot(x=pca_df["PC1"], y=pca_df["PC2"], hue=pca_df["anomaly"], pal
    plt.title(title)
    plt.xlabel("Principal Component 1")
    plt.ylabel("Principal Component 2")
```

```
    plt.legend(title="Anomaly", labels=["Normal", "Anomaly"])
    plt.show()
```

In [ ]:
```
plot_pca_2d(saw_pca_df, "Cylinder Bottom Saw - PCA (2D Visualization)")
```



In [ ]:
```
plot_pca_2d(lathe_pca_df, "CNC Lathe (Piston Rod) - PCA (2D Visualization)")
```



In [ ]:
```
joblib.dump(pca_saw, "saw_pca_model.pkl")
```

Out[ ]:
```
['saw_pca_model.pkl']
```

In [ ]:
```
joblib.dump(saw_scaler, "saw_scaler.pkl")
```

Out[ ]:
```
['saw_scaler.pkl']
```

In [ ]:
```
joblib.dump(pca_lathe, "lathe_pca_model.pkl")
```

Out[ ]:
```
['lathe_pca_model.pkl']
```

In [ ]:
```
joblib.dump(lathe_scaler, "lathe_scaler.pkl")
```

Out[ ]:
```
['lathe_scaler.pkl']
```

In [ ]:
```
def plot_pca_3d(pca_df, title):
    """
    Plots a 3D scatter plot of the first three principal components.

    Args:
        pca_df (pd.DataFrame): PCA-transformed DataFrame containing "PC1", "PC2"
        title (str): Title of the plot.
    """
    fig = px.scatter_3d(pca_df, x="PC1", y="PC2", z="PC3", color=pca_df["anomaly
                        color_discrete_map={"0": "blue", "1": "red"},
                        title=title, opacity=0.7)
    fig.show()
```

In [ ]:
```
plot_pca_3d(saw_pca_df, "Cylinder Bottom Saw - PCA (3D Visualization)")
```

In [ ]:
```
plot_pca_3d(lathe_pca_df, "CNC Lathe (Piston Rod) - PCA (3D Visualization)")
```

In [ ]:
```
from tensorflow.keras.models import Model
```

In [ ]:
```
from tensorflow.keras.layers import Input, Dense
```

In [ ]:
```
def build_autoencoder(input_dim):
    """
    Constructs an Autoencoder Model with an Encoder-Decoder architecture.

    Args:
        input_dim (int): Number of input features (PCA components).

    Returns:
        Model: Compiled Autoencoder model.
    """

    # Encoder
    input_layer = Input(shape=(input_dim,))
    encoded = Dense(32, activation="relu")(input_layer)
    encoded = Dense(16, activation="relu")(encoded)
    encoded = Dense(8, activation="relu")(encoded)

    # Decoder
    decoded = Dense(16, activation="relu")(encoded)
    decoded = Dense(32, activation="relu")(decoded)
    output_layer = Dense(input_dim, activation="linear")(decoded)

    # Autoencoder Model
    autoencoder = Model(input_layer, output_layer)

    # Compile Model
```

```
    autoencoder.compile(optimizer="adam", loss="mse")

    return autoencoder
```

In [ ]:
```
saw_train = saw_pca_df[saw_pca_df["anomaly"] == 0].drop(columns=["anomaly", "tim
```

In [ ]:
```
lathe_train = lathe_pca_df[lathe_pca_df["anomaly"] == 0].drop(columns=["anomaly"
```

In [ ]:
```
saw_input_dim = saw_train.shape[1]
```

In [ ]:
```
lathe_input_dim = lathe_train.shape[1]
```

In [ ]:
```
saw_autoencoder = build_autoencoder(saw_input_dim)
```

In [ ]:
```
lathe_autoencoder = build_autoencoder(lathe_input_dim)
```

In [ ]:
```
EPOCHS = 50
```

In [ ]:
```
BATCH_SIZE = 32
```

In [ ]:
```
print("🚀 Training Autoencoder for Cylinder Bottom Saw...")
```

```
🚀 Training Autoencoder for Cylinder Bottom Saw...
```

In [ ]:
```
saw_autoencoder.fit(saw_train, saw_train, epochs=EPOCHS, batch_size=BATCH_SIZE,
```

```
Epoch 1/50
8905/8905 [==============================] - 20s 2ms/step - loss: 0.3413 - val_lo
ss: 0.4552
Epoch 2/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.1952 - val_lo
ss: 0.3680
Epoch 3/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.1695 - val_lo
ss: 0.3358
Epoch 4/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.1540 - val_lo
ss: 0.3437
Epoch 5/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.1498 - val_lo
ss: 0.3709
Epoch 6/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.1324 - val_lo
ss: 0.3946
Epoch 7/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.1116 - val_lo
ss: 0.3299
Epoch 8/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.1042 - val_lo
ss: 0.2531
Epoch 9/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0976 - val_lo
ss: 0.1914
Epoch 10/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0864 - val_lo
ss: 0.1702
Epoch 11/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0826 - val_lo
ss: 0.1518
Epoch 12/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0811 - val_lo
ss: 0.1595
Epoch 13/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0722 - val_lo
ss: 0.1679
Epoch 14/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0732 - val_lo
ss: 0.1761
Epoch 15/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0639 - val_lo
ss: 0.1674
Epoch 16/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0757 - val_lo
ss: 0.1386
Epoch 17/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0773 - val_lo
ss: 0.2276
Epoch 18/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0674 - val_lo
ss: 0.1875
Epoch 19/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0639 - val_lo
ss: 0.2320
Epoch 20/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0621 - val_lo
ss: 0.2368
```

```
Epoch 21/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0659 - val_lo
ss: 0.1201
Epoch 22/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0631 - val_lo
ss: 0.1570
Epoch 23/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0596 - val_lo
ss: 0.1623
Epoch 24/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0654 - val_lo
ss: 0.1524
Epoch 25/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0639 - val_lo
ss: 0.1387
Epoch 26/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0638 - val_lo
ss: 0.1585
Epoch 27/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0574 - val_lo
ss: 0.1557
Epoch 28/50
8905/8905 [==============================] - 19s 2ms/step - loss: 0.0622 - val_lo
ss: 0.1887
Epoch 29/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0592 - val_lo
ss: 0.1669
Epoch 30/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0562 - val_lo
ss: 0.1496
Epoch 31/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0683 - val_lo
ss: 0.1301
Epoch 32/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0492 - val_lo
ss: 0.1506
Epoch 33/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0596 - val_lo
ss: 0.1415
Epoch 34/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0616 - val_lo
ss: 0.1368
Epoch 35/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0587 - val_lo
ss: 0.1396
Epoch 36/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0583 - val_lo
ss: 0.1647
Epoch 37/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0609 - val_lo
ss: 0.1507
Epoch 38/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0566 - val_lo
ss: 0.1403
Epoch 39/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0627 - val_lo
ss: 0.1475
Epoch 40/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0637 - val_lo
ss: 0.1273
```

```
Epoch 41/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0525 - val_lo
ss: 0.1217
Epoch 42/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0526 - val_lo
ss: 0.1161
Epoch 43/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0538 - val_lo
ss: 0.1393
Epoch 44/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0553 - val_lo
ss: 0.1402
Epoch 45/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0499 - val_lo
ss: 0.1190
Epoch 46/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0484 - val_lo
ss: 0.2764
Epoch 47/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0487 - val_lo
ss: 0.1312
Epoch 48/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0613 - val_lo
ss: 0.1106
Epoch 49/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0547 - val_lo
ss: 0.1298
Epoch 50/50
8905/8905 [==============================] - 18s 2ms/step - loss: 0.0518 - val_lo
ss: 0.1003
```

Out[ ]: <keras.callbacks.History at 0x1f08e5b9dc0>

In [ ]:
```
print("🚀 Training Autoencoder for CNC Lathe...")
```

```
🚀 Training Autoencoder for CNC Lathe...
```

In [ ]:
```
lathe_autoencoder.fit(lathe_train, lathe_train, epochs=EPOCHS, batch_size=BATCH_
```

```
Epoch 1/50
12716/12716 [==============================] - 28s 2ms/step - loss: 0.4474 - val_
loss: 0.3134
Epoch 2/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.3174 - val_
loss: 0.2872
Epoch 3/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2946 - val_
loss: 0.2781
Epoch 4/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2788 - val_
loss: 0.2594
Epoch 5/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2677 - val_
loss: 0.2558
Epoch 6/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2594 - val_
loss: 0.2512
Epoch 7/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2545 - val_
loss: 0.2434
Epoch 8/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2508 - val_
loss: 0.2396
Epoch 9/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2484 - val_
loss: 0.2366
Epoch 10/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2463 - val_
loss: 0.2354
Epoch 11/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2448 - val_
loss: 0.2356
Epoch 12/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2433 - val_
loss: 0.2332
Epoch 13/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2419 - val_
loss: 0.2289
Epoch 14/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2407 - val_
loss: 0.2315
Epoch 15/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2400 - val_
loss: 0.2376
Epoch 16/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2394 - val_
loss: 0.2264
Epoch 17/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2387 - val_
loss: 0.2280
Epoch 18/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2380 - val_
loss: 0.2300
Epoch 19/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2380 - val_
loss: 0.2334
Epoch 20/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2371 - val_
loss: 0.2281
```

```
Epoch 21/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2370 - val_
loss: 0.2270
Epoch 22/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2363 - val_
loss: 0.2258
Epoch 23/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2361 - val_
loss: 0.2382
Epoch 24/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2358 - val_
loss: 0.2264
Epoch 25/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2354 - val_
loss: 0.2235
Epoch 26/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2350 - val_
loss: 0.2240
Epoch 27/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2352 - val_
loss: 0.2235
Epoch 28/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2347 - val_
loss: 0.2260
Epoch 29/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2346 - val_
loss: 0.2232
Epoch 30/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2344 - val_
loss: 0.2270
Epoch 31/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2340 - val_
loss: 0.2246
Epoch 32/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2338 - val_
loss: 0.2258
Epoch 33/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2336 - val_
loss: 0.2295
Epoch 34/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2335 - val_
loss: 0.2215
Epoch 35/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2330 - val_
loss: 0.2229
Epoch 36/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2328 - val_
loss: 0.2226
Epoch 37/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2327 - val_
loss: 0.2287
Epoch 38/50
12716/12716 [==============================] - 28s 2ms/step - loss: 0.2324 - val_
loss: 0.2214
Epoch 39/50
12716/12716 [==============================] - 28s 2ms/step - loss: 0.2322 - val_
loss: 0.2242
Epoch 40/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2321 - val_
loss: 0.2222
```

```
Epoch 41/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2319 - val_
loss: 0.2217
Epoch 42/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2318 - val_
loss: 0.2230
Epoch 43/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2314 - val_
loss: 0.2215
Epoch 44/50
12716/12716 [==============================] - 28s 2ms/step - loss: 0.2312 - val_
loss: 0.2221
Epoch 45/50
12716/12716 [==============================] - 28s 2ms/step - loss: 0.2312 - val_
loss: 0.2218
Epoch 46/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2312 - val_
loss: 0.2230
Epoch 47/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2309 - val_
loss: 0.2200
Epoch 48/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2308 - val_
loss: 0.2182
Epoch 49/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2306 - val_
loss: 0.2209
Epoch 50/50
12716/12716 [==============================] - 27s 2ms/step - loss: 0.2303 - val_
loss: 0.2211
```

Out[ ]: <keras.callbacks.History at 0x1f059e88c10>

In [ ]:
```
# Since we have labeled anomalies, we can train a supervised Random Forest Class
```

In [ ]:
```
# Since we have labeled anomalies, we can train a supervised Random Forest Class
```

In [ ]:
```
# Since we have labeled anomalies, we can train a supervised Random Forest Class
```

In [ ]:
```
def compute_reconstruction_error(autoencoder, df):
    """
    Computes the reconstruction error for given data using the trained Autoencod

    Args:
        autoencoder (Model): Trained Autoencoder model.
        df (DataFrame): PCA-transformed data.

    Returns:
        np.array: Reconstruction errors.
    """
    reconstructed_data = autoencoder.predict(df)
    errors = np.mean(np.square(df - reconstructed_data), axis=1)
    return errors
```

In [ ]:
```
saw_reconstruction_errors = compute_reconstruction_error(saw_autoencoder, saw_pc
```

```
10402/10402 [==============================] - 14s 1ms/step
```

In [ ]:
```
lathe_reconstruction_errors = compute_reconstruction_error(lathe_autoencoder, la
```

```
15776/15776 [==============================] - 22s 1ms/step
```

In [ ]:
```
saw_pca_df["reconstruction_error"] = saw_reconstruction_errors
```

In [ ]:
```
lathe_pca_df["reconstruction_error"] = lathe_reconstruction_errors
```

In [ ]:
```
saw_threshold = np.percentile(saw_reconstruction_errors[saw_pca_df["anomaly"] ==
```

In [ ]:
```
lathe_threshold = np.percentile(lathe_reconstruction_errors[lathe_pca_df["anomal
```

In [ ]:
```
saw_pca_df["predicted_anomaly"] = (saw_pca_df["reconstruction_error"] > saw_thre
```

In [ ]:
```
lathe_pca_df["predicted_anomaly"] = (lathe_pca_df["reconstruction_error"] > lath
```

In [ ]:
```
from sklearn.metrics import classification_report, confusion_matrix
```

In [ ]:
```
print(" 🔹 Cylinder Bottom Saw - Autoencoder Performance")
```

```
 🔹 Cylinder Bottom Saw - Autoencoder Performance
```

In [ ]:
```
print(classification_report(saw_pca_df["anomaly"], saw_pca_df["predicted_anomaly
```

```
              precision    recall  f1-score   support

           0       0.95      0.95      0.95    316623
           1       0.09      0.09      0.09     16224

    accuracy                           0.91    332847
   macro avg       0.52      0.52      0.52    332847
weighted avg       0.91      0.91      0.91    332847
```

In [ ]:
```
print("\n 🔹 CNC Lathe - Autoencoder Performance")
```

```

 🔹 CNC Lathe - Autoencoder Performance
```

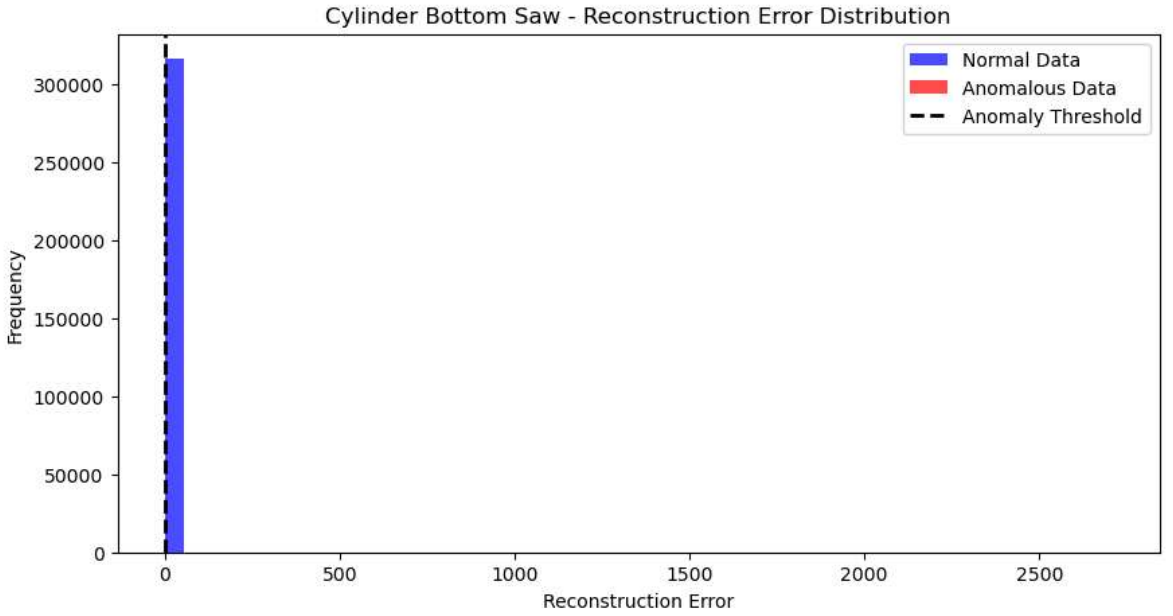
In [ ]:
```
print(classification_report(lathe_pca_df["anomaly"], lathe_pca_df["predicted_anc
```

```
              precision    recall  f1-score   support

           0       0.90      0.95      0.92    452115
           1       0.10      0.05      0.07     52711

    accuracy                           0.86    504826
   macro avg       0.50      0.50      0.49    504826
weighted avg       0.81      0.86      0.83    504826
```

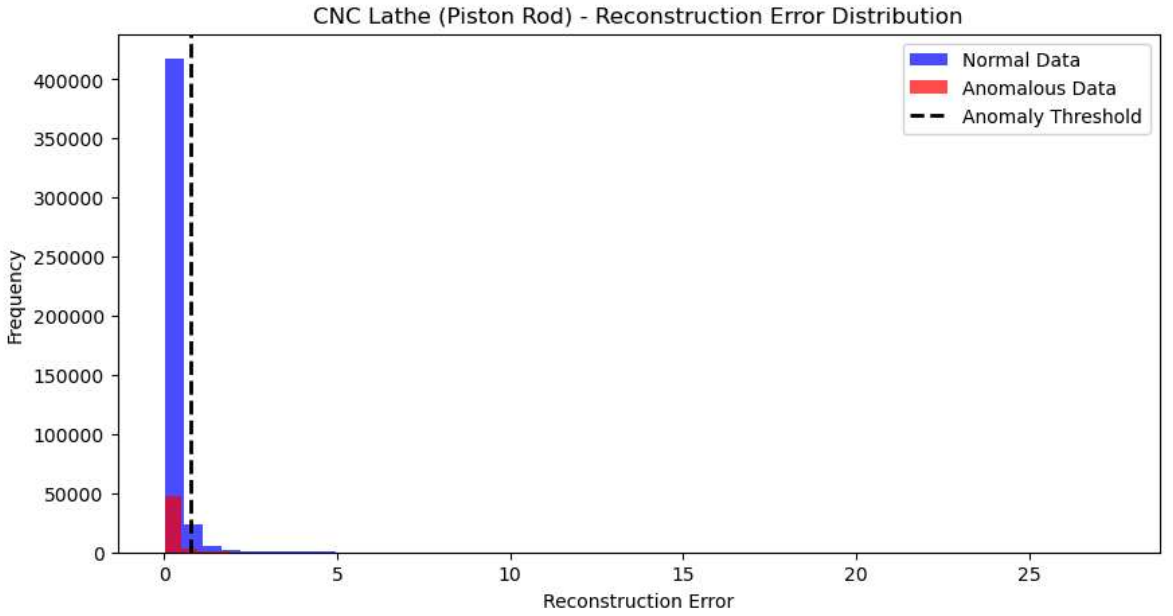
In [ ]:
```
def plot_reconstruction_error(df, threshold, title):
    """
    Plots a histogram of reconstruction errors.

    Args:
        df (DataFrame): The dataset containing reconstruction errors and anomaly
        threshold (float): The anomaly detection threshold.
        title (str): Title of the plot.
    """
    plt.figure(figsize=(10, 5))
    plt.hist(df[df["anomaly"] == 0]["reconstruction_error"], bins=50, alpha=0.7,
    plt.hist(df[df["anomaly"] == 1]["reconstruction_error"], bins=50, alpha=0.7,
```

```
    plt.axvline(threshold, color="black", linestyle="dashed", linewidth=2, label
    plt.xlabel("Reconstruction Error")
    plt.ylabel("Frequency")
    plt.title(title)
    plt.legend()
    plt.show()
```

In [ ]:
```
plot_reconstruction_error(saw_pca_df, saw_threshold, "Cylinder Bottom Saw - Reco
```



In [ ]:
```
plot_reconstruction_error(lathe_pca_df, lathe_threshold, "CNC Lathe (Piston Rod)
```



In [ ]:
```
from sklearn.svm import OneClassSVM
```

In [ ]:
```
saw_train_normal = saw_pca_df[saw_pca_df["anomaly"] == 0]["reconstruction_error"
```

In [ ]:
```
lathe_train_normal = lathe_pca_df[lathe_pca_df["anomaly"] == 0]["reconstruction_
```

In [ ]:
```
saw_svm = OneClassSVM(kernel="rbf", gamma="scale", nu=0.01)  # nu controls anoma
```

In [ ]:
```
lathe_svm = OneClassSVM(kernel="rbf", gamma="scale", nu=0.01)
```

In [ ]:
```
saw_svm.fit(saw_train_normal)
```

Out[ ]:
```
OneClassSVM
OneClassSVM(nu=0.01)
```

In [ ]:
```
lathe_svm.fit(lathe_train_normal)
```

Out[ ]:
```
OneClassSVM
OneClassSVM(nu=0.01)
```

In [ ]:
```
print("✅ One-Class SVM trained on normal reconstruction errors.")
```

```
✅ One-Class SVM trained on normal reconstruction errors.
```

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [ ]:
```
from sklearn.model_selection import train_test_split
```

In [ ]:
```
saw_X = saw_pca_df.drop(columns=["anomaly", "timestamp", "reconstruction_error",
```

In [ ]:
```
saw_y = saw_pca_df["anomaly"]
```

In [ ]:
```
lathe_X = lathe_pca_df.drop(columns=["anomaly", "timestamp", "reconstruction_err
```

In [ ]:
```
lathe_y = lathe_pca_df["anomaly"]
```

In [ ]:
```
saw_X_train, saw_X_test, saw_y_train, saw_y_test = train_test_split(saw_X, saw_y
```

In [ ]:
```
lathe_X_train, lathe_X_test, lathe_y_train, lathe_y_test = train_test_split(lath
```

In [ ]:
```
saw_rf = RandomForestClassifier(n_estimators=100, random_state=42)
```

In [ ]:
```
lathe_rf = RandomForestClassifier(n_estimators=100, random_state=42)
```

In [ ]:
```
saw_rf.fit(saw_X_train, saw_y_train)
```

Out[ ]:
```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

In [ ]:
```
lathe_rf.fit(lathe_X_train, lathe_y_train)
```

Out[ ]:
```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

In [ ]:
```
print("✅ Random Forest Classifier trained on labeled data.")
```

✅ Random Forest Classifier trained on labeled data.

In [ ]:
```
saw_pca_df["svm_predicted_anomaly"] = saw_svm.predict(saw_pca_df["reconstruction
```

In [ ]:
```
lathe_pca_df["svm_predicted_anomaly"] = lathe_svm.predict(lathe_pca_df["reconstr
```

In [ ]:
```
saw_pca_df["svm_predicted_anomaly"] = saw_pca_df["svm_predicted_anomaly"].apply(
```

In [ ]:
```
lathe_pca_df["svm_predicted_anomaly"] = lathe_pca_df["svm_predicted_anomaly"].ap
```

In [ ]:
```
from sklearn.metrics import classification_report
```

In [ ]:
```
print(" 🔹 Cylinder Bottom Saw - One-Class SVM Performance")
```

🔹 Cylinder Bottom Saw - One-Class SVM Performance

In [ ]:
```
print(classification_report(saw_pca_df["anomaly"], saw_pca_df["svm_predicted_anc
```

```
              precision    recall  f1-score   support

           0       0.95      0.99      0.97    316623
           1       0.02      0.00      0.01     16224

    accuracy                           0.94    332847
   macro avg       0.48      0.50      0.49    332847
weighted avg       0.91      0.94      0.92    332847
```

In [ ]:
```
print("\n 🔹 CNC Lathe - One-Class SVM Performance")
```

🔹 CNC Lathe - One-Class SVM Performance

In [ ]:
```
print(classification_report(lathe_pca_df["anomaly"], lathe_pca_df["svm_predicted
```

```
              precision    recall  f1-score   support

           0       0.90      0.99      0.94    452115
           1       0.10      0.01      0.02     52711

    accuracy                           0.89    504826
   macro avg       0.50      0.50      0.48    504826
weighted avg       0.81      0.89      0.84    504826
```

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [ ]:
```
from sklearn.model_selection import train_test_split
```

In [ ]:
```
saw_X = saw_pca_df.drop(columns=["anomaly", "timestamp", "reconstruction_error",
```

In [ ]:
```
saw_y = saw_pca_df["anomaly"]
```

In [ ]:
```
saw_pca_df["rf_predicted_anomaly"] = saw_rf.predict(saw_X)
```

In [ ]:
```
lathe_pca_df["rf_predicted_anomaly"] = lathe_rf.predict(lathe_X)
```

In [ ]:
```
print(" 🔹 Cylinder Bottom Saw - Random Forest Performance")
```

◆ Cylinder Bottom Saw - Random Forest Performance

In [ ]:
```
print(classification_report(saw_y, saw_pca_df["rf_predicted_anomaly"]))
```

```
              precision    recall  f1-score   support

           0       0.98      0.99      0.99    316623
           1       0.72      0.69      0.71     16224

    accuracy                           0.97    332847
   macro avg       0.85      0.84      0.85    332847
weighted avg       0.97      0.97      0.97    332847
```

In [ ]:
```
print("\n ◆  CNC Lathe - Random Forest Performance")
```

◆ CNC Lathe - Random Forest Performance

In [ ]:
```
print(classification_report(lathe_y, lathe_pca_df["rf_predicted_anomaly"]))
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      0.99    452115
           1       1.00      0.89      0.94     52711

    accuracy                           0.99    504826
   macro avg       0.99      0.95      0.97    504826
weighted avg       0.99      0.99      0.99    504826
```

In [ ]:
```
def majority_vote(df, model_columns):
    """
    Performs majority voting across multiple anomaly detection models.

    Args:
        df (DataFrame): Data containing anomaly predictions.
        model_columns (list): List of model prediction columns.

    Returns:
        pd.Series: Final ensemble predictions.
    """
    return df[model_columns].mode(axis=1)[0]
```

In [ ]:
```
saw_pca_df["ensemble_predicted_anomaly"] = majority_vote(saw_pca_df, ["predicted
```

In [ ]:
```
lathe_pca_df["ensemble_predicted_anomaly"] = majority_vote(lathe_pca_df, ["predi
```

In [ ]:
```
print(" ◆  Cylinder Bottom Saw - Ensemble Model Performance")
```

◆ Cylinder Bottom Saw - Ensemble Model Performance

In [ ]:
```
print(classification_report(saw_y, saw_pca_df["ensemble_predicted_anomaly"]))
```

	precision	recall	f1-score	support
0	0.95	1.00	0.97	316623
1	0.47	0.06	0.11	16224
accuracy			0.95	332847
macro avg	0.71	0.53	0.54	332847
weighted avg	0.93	0.95	0.93	332847

In [ ]:
```
print("\n 🔹 CNC Lathe - Ensemble Model Performance")
```

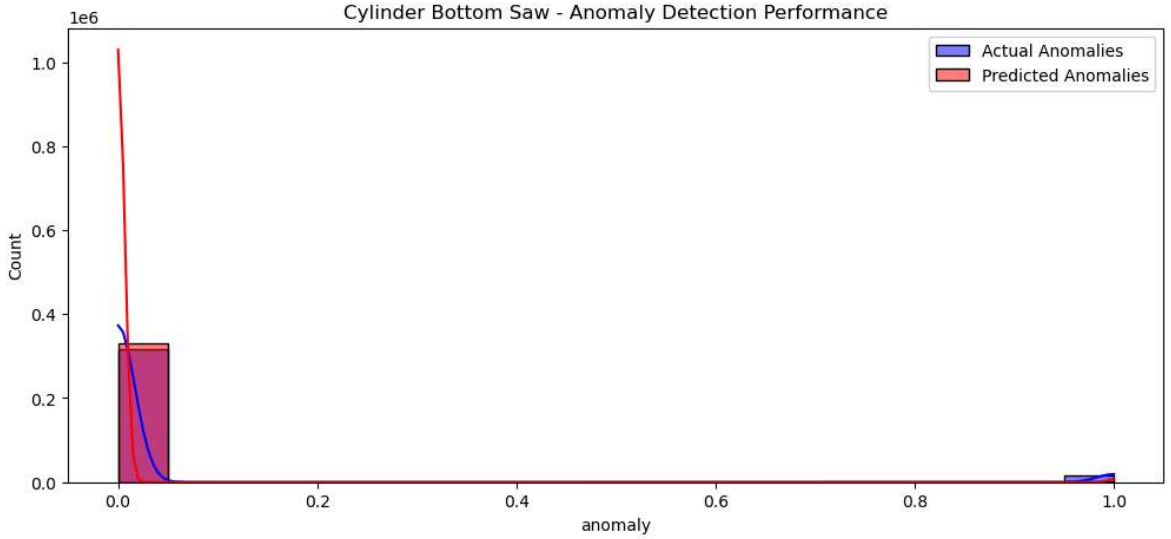
```
 🔹 CNC Lathe - Ensemble Model Performance
```

In [ ]:
```
print(classification_report(lathe_y, lathe_pca_df["ensemble_predicted_anomaly"])
```

	precision	recall	f1-score	support
0	0.90	0.99	0.94	452115
1	0.39	0.04	0.08	52711
accuracy			0.89	504826
macro avg	0.64	0.52	0.51	504826
weighted avg	0.85	0.89	0.85	504826

In [ ]:
```
def plot_anomaly_comparison(df, title):
    """
    Plots actual vs. predicted anomalies.

    Args:
        df (DataFrame): Data containing actual and predicted anomalies.
        title (str): Title of the plot.
    """
    plt.figure(figsize=(12, 5))
    sns.histplot(df["anomaly"], label="Actual Anomalies", color="blue", kde=True
    sns.histplot(df["ensemble_predicted_anomaly"], label="Predicted Anomalies",
    plt.title(title)
    plt.legend()
    plt.show()
```

In [ ]:
```
plot_anomaly_comparison(saw_pca_df, "Cylinder Bottom Saw - Anomaly Detection Per
```



In [ ]:
```
plot_anomaly_comparison(lathe_pca_df, "CNC Lathe - Anomaly Detection Performance
```