Wind Turbine Structural Health Monitoring – We Do Wind

Chetan Chandra Konda 12504278 [1], Akhil Goud Chinthakindi 12503777 [2], Nikunj Mistry 12403425 [3], and Zaake Enock 12504721 [4]

[1,2,3,4]Master of Engineering: Applied AI for Digital Production and Management, Technische Hochschule Deggendorf, Cham, Germany

July 3, 2025

Abstract

This document presents a comprehensive analysis of wind turbine structural health monitoring using advanced machine learning techniques. The study focuses on detecting and identifying various operational conditions such as rotor icing, pitch drive failure, and aerodynamic imbalance. By leveraging time-series data from the Aventa AV-7 Research Wind Turbine, we employ advanced machine learning techniques, including LSTM Autoencoders and CNN+LSTM models, to capture spatial and temporal features.

1 Introduction

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

2 Data Collection

2.1 Data Sources

The data used in this study were collected from the Aventa AV-7 Research Wind Turbine located in Taggenberg and owned by the ETH Zurich Department of Structural Health Monitoring. The data consists of time-series measurements stored in HDF5 format, capturing various operational parameters of the wind turbine.

2.2 Conditions and Time Periods

- **Rotor Icing:** Data collected from September 3, 2022, to December 20, 2022.

- **Pitch Drive Failure:** Data collected from January 22, 2022, to February 27, 2022.
- **Aerodynamic Imbalance:** Data collected from September 3, 2022, to January 21, 2023.

2.3 Signals Collected

The dataset includes several key operational parameters:

- Wind Speed (WM1)
- Power Output (WM2)
- Rotor Speed (WM3)
- Temperature (ATM_TEMP_01)
- Rotor Acceleration (GEN_ACC_XX_01)

3 Data Preprocessing

3.1 Loading and Extracting Data

The initial step involves loading the HDF5 files and extracting the relevant signals. This process ensures that we have a clean and structured dataset for further analysis.

3.2 Handling Missing Values

Missing values in the dataset are filled with zeros to ensure data consistency. This step is crucial for maintaining the integrity of the dataset and preventing biases in the analysis.

3.3 Timestamp Formatting

Timestamps are formatted to a standard datetime format to facilitate time-series analysis. This step ensures that the temporal aspect of the data is correctly captured and utilized.

3.4 Normalization

Features are normalized to a range of [0, 1] using Min-Max scaling. Normalization ensures that all features contribute equally to the model and helps in faster convergence during training.

4 Exploratory Data Analysis (EDA)

4.1 Visualizations

Various visualizations are used to explore the dataset and gain insights into the data:

- Box Plots
- Line Plots
- Distribution Plots
- Correlation Matrix

4.2 Outlier Detection

Outliers in the data are identified and addressed to prevent skewing the model. This step ensures that the model is trained on a representative dataset.

4.3 Trend Analysis

Trends and patterns over time are observed to understand the temporal dynamics of the data. This analysis helps in identifying key operational conditions and anomalies.

4.4 Feature Relationships

Correlations between different signals are analyzed to understand their relationships. This analysis aids in feature selection and model interpretation.

5 Feature Engineering

5.1 Rolling Statistics

Rolling mean and standard deviation are calculated for each feature to capture temporal trends and variability in the data. These statistics provide additional insights into the temporal dynamics of the operational parameters.

5.2 Derived Features

New features are derived from the original features to capture more complex patterns and relationships. Examples include:

- Wind Power Ratio
- Temperature Difference

5.3 Sequence Creation

The time-series data is converted into sequences of fixed length for training sequential models like LSTMs. This step ensures that the temporal dependencies in the data are captured and utilized by the model.

6 Model Architecture

6.1 LSTM Autoencoder

The LSTM Autoencoder is used for anomaly detection by learning to reconstruct the input data. The architecture consists of an encoder-decoder structure with LSTM layers. The encoder compresses the input sequence into a context vector, and the decoder reconstructs the original sequence from this context vector.

6.2 CNN+LSTM Model

The CNN+LSTM model combines the strengths of Convolutional Neural Networks (CNNs) and Long Short-Term Memory networks (LSTMs) for classification tasks. The CNN layers extract spatial features from the input data, while the LSTM layers capture temporal dependencies. The final fully connected layers combine these features for classification.

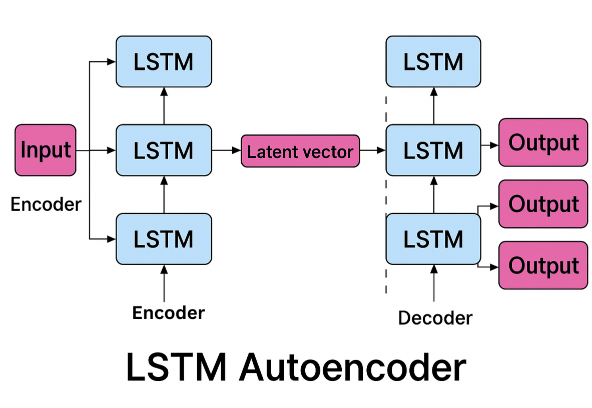


Figure 1: Structure of LSTM Autoencoder

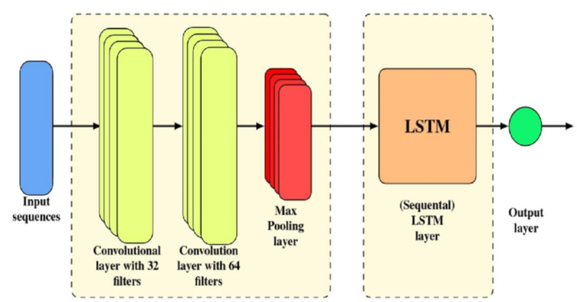


Figure 2: Structure of CNN+LSTM Model

7 Pitch Drive Failure Analysis

7.1 Distribution Plots

The distribution plots for the pitch drive dataset provide insights into the spread and range of each feature.

7.2 Correlation Matrix

The correlation matrix for the pitch drive dataset shows the relationships between different features.

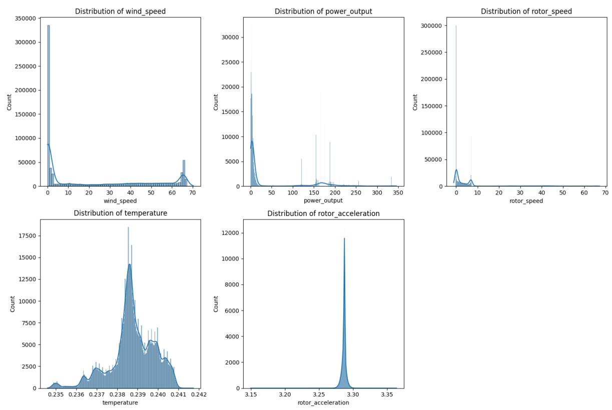


Figure 3: Distribution Plots for Pitch Drive Dataset

7.3 Box Plots

The box plots for the pitch drive dataset before preprocessing show the presence of outliers and the spread of each feature.

7.4 Training Steps

The training steps for both models on the pitch drive dataset are visualized in the following image.

7.5 Training and Validation Loss

The training and validation loss for the pitch drive dataset are shown in the following plot.

7.6 Model Evaluation

The evaluation metrics for the pitch drive dataset are presented in the following table.

Metric	Value
Test Loss (LSTM Autoencoder)	0.0071
Test Loss (CNN+LSTM)	0.02332

Table 1: Model Evaluation for Pitch Drive Dataset

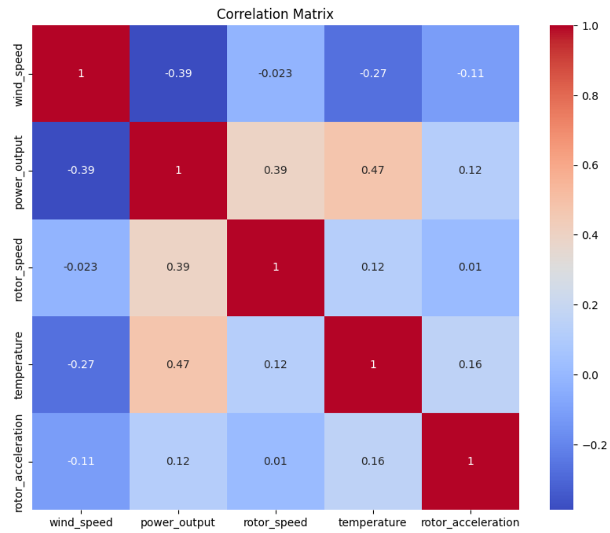


Figure 4: Correlation Matrix for Pitch Drive Dataset

8 Rotor Icing Analysis

8.1 Training and Validation Loss

The training and validation loss for the rotor icing dataset are shown in the following plot.

8.2 Model Evaluation

The evaluation metrics for the rotor icing dataset are presented in the following table.

Metric	Value
Test Loss (LSTM Autoencoder)	0.0066
Test Loss (CNN+LSTM)	0.0

Table 2: Model Evaluation for Rotor Icing Dataset

9 Conclusion

This study successfully developed and trained machine learning models to detect and identify various operational conditions in wind turbines. By leveraging advanced techniques such as LSTM Autoencoders and CNN+LSTM models, we captured both

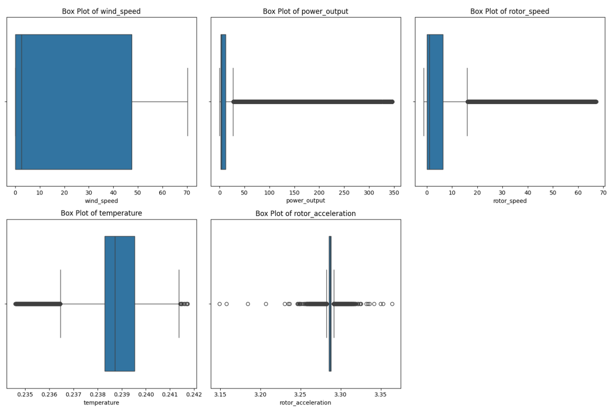
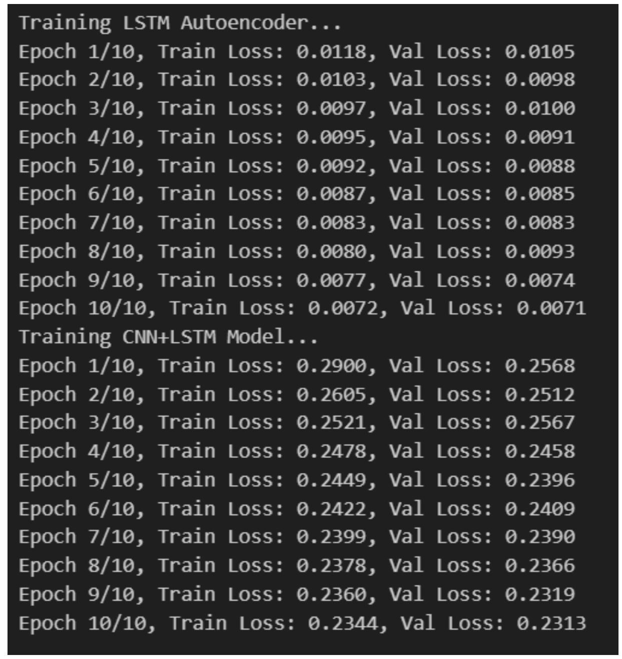


Figure 5: Box Plots for Pitch Drive Dataset Before Preprocessing

spatial and temporal features in the data, leading to improved maintenance strategies and operational efficiency. Future work could explore additional features and advanced models for better performance and more accurate predictions.



```
Training LSTM Autoencoder...
Epoch 1/10, Train Loss: 0.0118, Val Loss: 0.0105
Epoch 2/10, Train Loss: 0.0103, Val Loss: 0.0098
Epoch 3/10, Train Loss: 0.0097, Val Loss: 0.0100
Epoch 4/10, Train Loss: 0.0095, Val Loss: 0.0091
Epoch 5/10, Train Loss: 0.0092, Val Loss: 0.0088
Epoch 6/10, Train Loss: 0.0087, Val Loss: 0.0085
Epoch 7/10, Train Loss: 0.0083, Val Loss: 0.0083
Epoch 8/10, Train Loss: 0.0080, Val Loss: 0.0093
Epoch 9/10, Train Loss: 0.0077, Val Loss: 0.0074
Epoch 10/10, Train Loss: 0.0072, Val Loss: 0.0071
Training CNN+LSTM Model...
Epoch 1/10, Train Loss: 0.2900, Val Loss: 0.2568
Epoch 2/10, Train Loss: 0.2605, Val Loss: 0.2512
Epoch 3/10, Train Loss: 0.2521, Val Loss: 0.2567
Epoch 4/10, Train Loss: 0.2478, Val Loss: 0.2458
Epoch 5/10, Train Loss: 0.2449, Val Loss: 0.2396
Epoch 6/10, Train Loss: 0.2422, Val Loss: 0.2409
Epoch 7/10, Train Loss: 0.2399, Val Loss: 0.2390
Epoch 8/10, Train Loss: 0.2378, Val Loss: 0.2366
Epoch 9/10, Train Loss: 0.2360, Val Loss: 0.2319
Epoch 10/10, Train Loss: 0.2344, Val Loss: 0.2313
```

Figure 6: Training Steps for Pitch Drive Dataset

References

1. Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735-1780.

2. LeCun, Y., Bottou, L., Orr, G. B., & Müller, K. R. (1998). Efficient BackProp. In Neural Networks: Tricks of the Trade (pp. 9-48). Springer, Berlin, Heidelberg.

3. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

4. Chollet, F. (2017). Deep Learning with Python. Manning Publications.

5. Brownlee, J. (2017). Deep Learning for Time Series Forecasting. Machine Learning Mastery.

Appendix

1. **Dataset Link:** https://zenodo.org/records/82297

2. **Code Link (GitHub):** https://github.com/enockzaake/wedowind

Code

Data Preprocessing and Feature Engineering

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
from scipy.signal import detrend
from sklearn.preprocessing import MinMaxScaler

def extract_date(file_name):
    date_str = '_'.join(file_name.split('_')[-3:]).replace('.hdf5', '')
    return datetime.strptime(date_str, "%d_%m_%Y")

def extract_time_pd(time_str):
```

Figure 7: Training and Validation Loss for Pitch Drive Dataset

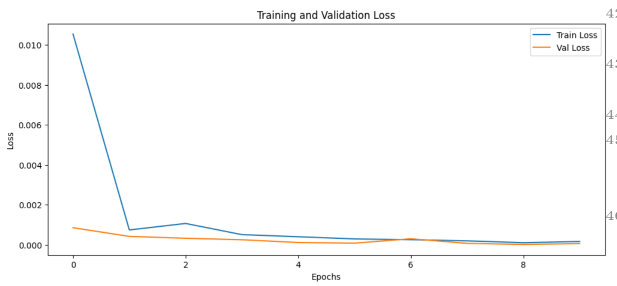


Figure 8: Training and Validation Loss for Rotor Icing Dataset

```python
    hours, minutes, seconds = map(int, time_str.split('_'))
    return pd.Timedelta(hours=hours, minutes=minutes, seconds=seconds)

def format_timestamp(file_date, time_stamp):
    time_delta = extract_time_pd(time_stamp)
    return file_date + time_delta

def calculate_rolling_mean(data, window_size):
    return data.rolling(window=window_size).mean()

def calculate_rolling_std(data, window_size):
    return data.rolling(window=window_size).std()

def process_hdf5_files(base_path, file_names, window_size=10):
    sorted_file_names = sorted(file_names, key=extract_date)
    all_features = {}

    for file_name in sorted_file_names:
        file_path = os.path.join(base_path, file_name)
        features_data = []
        file_date = extract_date(file_name)

        with h5py.File(file_path, "r") as f:
            print(f"Processing file: {file_path}")
            for dataset_name in f.keys():
                for time_stamp in f[dataset_name].keys():
                    signal_list = list(f[dataset_name][time_stamp].keys())

                    if 'ChannelList' in signal_list:
                        signal_list.remove('ChannelList')

                    if not all(signal in signal_list for signal in required_signals):
                        print(f"Skipping timestamp {time_stamp} - required signal is missing")
                        continue

                    wind_speed = f[dataset_name][time_stamp]['WM1']['Value'][()].flatten()
                    power_output = f[dataset_name][time_stamp]['WM2']['Value'][()].flatten()
                    rotor_speed = f[dataset_name][time_stamp]['WM3']['Value'][()].flatten()
                    temperature = f[dataset_name][time_stamp]['ATM_TEMP_01']['Value'][()].flatten()
                    rotor_acceleration = f[dataset_name][time_stamp]['GEN_ACC_XX_01']['Value'][()].flatten()

                    if len(wind_speed) == 0 or len(power_output) == 0 or len(rotor_speed) == 0 or len(temperature) == 0 or len(rotor_acceleration) == 0:
                        print(f"Skipping timestamp {time_stamp} - one or more signals are empty")
                        continue

                    min_length = min(len(wind_speed), len(power_output), len(rotor_speed), len(temperature), len(
```

```python
rotor_acceleration))
                wind_speed = wind_speed[:min_length]
                power_output = power_output[:min_length]
                rotor_speed = rotor_speed[:min_length]
                temperature = temperature[:min_length]
                rotor_acceleration = rotor_acceleration[:min_length]

                wind_power_ratio = np.zeros(min_length)
                np.divide(power_output, wind_speed, out=wind_power_ratio, where=wind_speed != 0)

                temp_diff = np.diff(temperature)
                temp_diff = np.append(temp_diff, 0)

                wind_speed_series = pd.Series(wind_speed)
                power_output_series = pd.Series(power_output)
                rotor_speed_series = pd.Series(rotor_speed)
                temperature_series = pd.Series(temperature)

                wind_speed_rolling_mean = calculate_rolling_mean(wind_speed_series, window_size)
                wind_speed_rolling_std = calculate_rolling_std(wind_speed_series, window_size)
                power_output_rolling_mean = calculate_rolling_mean(power_output_series, window_size)
                power_output_rolling_std = calculate_rolling_std(power_output_series, window_size)
                rotor_speed_rolling_mean = calculate_rolling_mean(rotor_speed_series, window_size)
                rotor_speed_rolling_std = calculate_rolling_std(rotor_speed_series, window_size)
                temperature_rolling_mean = calculate_rolling_mean(temperature_series, window_size)
                temperature_rolling_std = calculate_rolling_std(temperature_series, window_size)

                formatted_timestamp = format_timestamp(file_date, time_stamp)

                features = pd.DataFrame({
                    'timestamp': formatted_timestamp,
                    'wind_speed': wind_speed,
                    'power_output': power_output,
                    'rotor_speed': rotor_speed,
                    'temperature': temperature,
                    'wind_power_ratio': wind_power_ratio,
                    'rotor_acceleration': rotor_acceleration,
                    'temp_diff': temp_diff,
                    'wind_speed_rolling_mean': wind_speed_rolling_mean,
                    'wind_speed_rolling_std': wind_speed_rolling_std,
                    'power_output_rolling_mean': power_output_rolling_mean,
                    'power_output_rolling_std': power_output_rolling_std,
                    'rotor_speed_rolling_mean': rotor_speed_rolling_mean,
                    'rotor_speed_rolling_std': rotor_speed_rolling_std,
                    'temperature_rolling_mean': temperature_rolling_mean,
                    'temperature_rolling_std': temperature_rolling_std
                })

                features_data.append(features)

    if features_data:
        all_features[file_name] = pd.concat(features_data, ignore_index=True)
    else:
        all_features[file_name] = pd.DataFrame()
```

```python

    return all_features

def save_features_to_csv(data, output_dir, file_name):
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    combined_df = pd.concat(data.values(), ignore_index=True)
    combined_csv_path = os.path.join(output_dir, f"{file_name}.csv")
    combined_df.to_csv(combined_csv_path, index=False)
    print(f"All features combined and saved to {combined_csv_path}")

def plot_eda(combined_df):
    plt.figure(figsize=(15, 10))
    for i, col in enumerate(target_signals, 1):
        plt.subplot(2, 3, i)
        sns.boxplot(x=combined_df[col])
        plt.title(f'Box Plot of {col}')
    plt.tight_layout()
    plt.savefig('box_plots.png')
    plt.show()

    plt.figure(figsize=(15, 10))
    for i, col in enumerate(target_signals, 1):
        plt.subplot(2, 3, i)
        sns.lineplot(x=combined_df.index, y=combined_df[col])
        plt.title(f'Line Plot of {col}')
    plt.tight_layout()
    plt.savefig('line_plots.png')
    plt.show()

    plt.figure(figsize=(15, 10))
    for i, col in enumerate(target_signals, 1):
        plt.subplot(2, 3, i)
        sns.histplot(combined_df[col], kde=True)
        plt.title(f'Distribution of {col}')
    plt.tight_layout()
    plt.savefig('distribution_plots.png')
    plt.show()

    plt.figure(figsize=(10, 8))
    sns.heatmap(combined_df[target_signals].corr(), annot=True, cmap='coolwarm')
    plt.title('Correlation Matrix')
    plt.savefig('correlation_matrix.png')
    plt.show()

aerodynamic_imbalance_file_names = [
    "Aventa_Taggenberg_08_04_2022.hdf5",
    "Aventa_Taggenberg_09_04_2022.hdf5",
    "Aventa_Taggenberg_07_08_2022.hdf5",
    "Aventa_Taggenberg_03_09_2022.hdf5",
    "Aventa_Taggenberg_01_11_2022.hdf5",
    "Aventa_Taggenberg_04_11_2022.hdf5",
    "Aventa_Taggenberg_08_12_2022.hdf5",
    "Aventa_Taggenberg_11_12_2022.hdf5",
    "Aventa_Taggenberg_19_12_2022.hdf5",
    "Aventa_Taggenberg_23_12_2022.hdf5",
    "Aventa_Taggenberg_29_12_2022.hdf5",
    "Aventa_Taggenberg_04_01_2023.hdf5",
    "Aventa_Taggenberg_15_01_2023.hdf5",
    "Aventa_Taggenberg_21_01_2023.hdf5",
]

base_path = r"datasets/raw_data/aerodynamic_imbalance/"
output_dir = r"datasets/"

target_signals = ["wind_speed", "power_output", "rotor_speed", "temperature", "rotor_acceleration"]

if __name__ == "__main__":
    all_features = process_hdf5_files(base_path, aerodynamic_imbalance_file_names)
    combined_df = pd.concat(all_features.values(), ignore_index=True)
    plot_eda(combined_df)
    save_features_to_csv(all_features, output_dir, "aerodynamic_imbalance")
```

Model Training and Evaluation

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

import torch
import torch.nn as nn
from torch.utils.data import DataLoader, TensorDataset

df = pd.read_csv("datasets/aerodynamic_imbalance.csv")
```

```python
df['timestamp'] = pd.to_datetime(df['timestamp'], format='%d-%m-%Y')
df['aerodynamic_imbalance'] = df['timestamp'].apply(lambda x: 1 if x.date() >= pd.to_datetime('2022-12-08').date() else 0)

target_signals = ["wind_speed", "power_output", "rotor_speed", "temperature", "rotor_acceleration"]

scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(df[target_signals])
y = df["aerodynamic_imbalance"].values.astype(np.float32)

def create_sequences(data, labels, window_size):
    sequences = []
    targets = []
    for i in range(len(data) - window_size):
        sequences.append(data[i:i + window_size])
        targets.append(labels[i + window_size])
    return np.array(sequences), np.array(targets)

SEQ_LEN = 20
X_seq, y_seq = create_sequences(X_scaled, y, SEQ_LEN)

X_train, X_test, y_train, y_test = train_test_split(X_seq, y_seq, test_size=0.2, random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.25, random_state=42)

X_train_t = torch.tensor(X_train, dtype=torch.float32)
X_val_t = torch.tensor(X_val, dtype=torch.float32)
X_test_t = torch.tensor(X_test, dtype=torch.float32)
y_train_t = torch.tensor(y_train, dtype=torch.float32)
y_val_t = torch.tensor(y_val, dtype=torch.float32)
y_test_t = torch.tensor(y_test, dtype=torch.float32)

train_dataset = TensorDataset(X_train_t, y_train_t)
val_dataset = TensorDataset(X_val_t, y_val_t)
test_dataset = TensorDataset(X_test_t, y_test_t)

batch_size = 64
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=batch_size)
test_loader = DataLoader(test_dataset, batch_size=batch_size)

class LSTMAutoencoder(nn.Module):
    def __init__(self, input_dim, hidden_dim):
        super().__init__()
        self.encoder = nn.LSTM(input_dim, hidden_dim, batch_first=True)
        self.decoder = nn.LSTM(hidden_dim, input_dim, batch_first=True)

    def forward(self, x):
        _, (h, _) = self.encoder(x)
        h = h.repeat(x.size(0), 1, 1)
        c = torch.zeros_like(h)
        decoded, _ = self.decoder(x, (h, c))
        return decoded

class CNNLSTMClassifier(nn.Module):
    def __init__(self, input_dim, hidden_dim):
        super().__init__()
        self.cnn = nn.Sequential(
            nn.Conv1d(in_channels=input_dim, out_channels=32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool1d(kernel_size=2)
        )
        self.lstm = nn.LSTM(input_size=32, hidden_size=hidden_dim, batch_first=True)
        self.fc = nn.Linear(hidden_dim, 1)

    def forward(self, x):
        x = x.permute(0, 2, 1)
        x = self.cnn(x)
        x = x.permute(0, 2, 1)
        _, (h, _) = self.lstm(x)
        return torch.sigmoid(self.fc(h[-1]))

input_dim = X_train.shape[2]
hidden_dim = 64

autoencoder = LSTMAutoencoder(input_dim=input_dim, hidden_dim=hidden_dim)
model = CNNLSTMClassifier(input_dim=input_dim, hidden_dim=hidden_dim)

criterion = nn.MSELoss()
```

```python
loss_fn = nn.BCELoss()
optimizer_ae = torch.optim.Adam(autoencoder.parameters(), lr=1e-3)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)

def train_autoencoder(model, train_loader, val_loader, criterion, optimizer, num_epochs):
    train_losses = []
    val_losses = []

    for epoch in range(num_epochs):
        model.train()
        train_loss = 0.0

        for xb, _ in train_loader:
            optimizer.zero_grad()
            outputs = model(xb)
            loss = criterion(outputs, xb)
            loss.backward()
            optimizer.step()
            train_loss += loss.item()

        train_loss /= len(train_loader)
        train_losses.append(train_loss)

        model.eval()
        val_loss = 0.0

        with torch.no_grad():
            for xb, _ in val_loader:
                outputs = model(xb)
                loss = criterion(outputs, xb)
                val_loss += loss.item()

        val_loss /= len(val_loader)
        val_losses.append(val_loss)

        print(f'Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss:.4f}, Val Loss: {val_loss:.4f}')

    return train_losses, val_losses

def train_model(model, train_loader, val_loader, criterion, optimizer, num_epochs):
    train_losses = []
    val_losses = []

    for epoch in range(num_epochs):
        model.train()
        train_loss = 0.0

        for xb, yb in train_loader:
            optimizer.zero_grad()
            outputs = model(xb).squeeze()
            loss = criterion(outputs, yb)
            loss.backward()
            optimizer.step()
            train_loss += loss.item()

        train_loss /= len(train_loader)
        train_losses.append(train_loss)

        model.eval()
        val_loss = 0.0

        with torch.no_grad():
            for xb, yb in val_loader:
                outputs = model(xb).squeeze()
                loss = criterion(outputs, yb)
                val_loss += loss.item()

        val_loss /= len(val_loader)
        val_losses.append(val_loss)

        print(f'Epoch {epoch+1}/{num_epochs}, Train Loss: {train_loss:.4f}, Val Loss: {val_loss:.4f}')

    return train_losses, val_losses

print("Training LSTM Autoencoder...")
train_losses_ae, val_losses_ae = train_autoencoder(autoencoder, train_loader, val_loader, criterion, optimizer_ae, num_epochs=10)

plt.figure(figsize=(12, 5))
plt.plot(train_losses_ae, label='Train Loss')
plt.plot(val_losses_ae, label='Val Loss')
plt.title('Training and Validation Loss for LSTM Autoencoder')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.savefig('lstm_loss.png')
plt.show()

print("Training CNN+LSTM Model...")
train_losses, val_losses = train_model(model, train_loader, val_loader, loss_fn, optimizer, num_epochs=10)

plt.figure(figsize=(12, 5))
plt.plot(train_losses, label='Train Loss')
plt.plot(val_losses, label='Val Loss')
plt.title('Training and Validation Loss for
```

```
    CNN+LSTM Model')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.savefig('cnn_lstm_loss.png')
plt.show()

def evaluate_model(model, test_loader, criterion):
    model.eval()
    test_loss = 0.0

    with torch.no_grad():
        for xb, yb in test_loader:
            outputs = model(xb).squeeze()
            loss = criterion(outputs, yb)
            test_loss += loss.item()

    test_loss /= len(test_loader)
    print(f'Test Loss: {test_loss:.4f}')
    return test_loss

print("Evaluating LSTM Autoencoder...")
lstm_test_loss = evaluate_model(autoencoder, test_loader, criterion)

print("Evaluating CNN+LSTM Model...")
cnn_lstm_test_loss = evaluate_model(model, test_loader, loss_fn)

torch.save(autoencoder.state_dict(), 'lstm_autoencoder_aerodynamic_imbalance.pth')
torch.save(model.state_dict(), 'cnn_lstm_model_aerodynamic_imbalance.pth')
print("Models saved to lstm_autoencoder_aerodynamic_imbalance.pth and cnn_lstm_model_aerodynamic_imbalance.pth")
```