ISI Project notes:

Created	@June 13, 2025 3:15 PM
Class	ISI
Event	Project
Subject	AI

Project Link:

https://colab.research.google.com/drive/1pLfXow-mBJZWR8OFnd5koFUgKUYZJXT9?usp=sharing

References:

FFT ref: https://youtu.be/nmgFG7PUHfo?feature=shared

Butterworth filter ref:

https://youtu.be/dmzikG1jZpU?feature=shared

https://www.youtube.com/watch?v=-EHRFrDujhc

https://youtu.be/hg3wMmTI_Oc?feature=shared

https://youtu.be/3yyp5JRqNXs?feature=shared

Empirical Rule (3 sigma rule) ref:

https://youtu.be/n7phemRMb98?feature=shared

Crest factor visualizer that we made:

https://www.perplexity.ai/apps/1a341aa6-7ce9-42da-9c04-24bb74b02d2c

Harmonics ref: https://youtu.be/EYRmB1aNh9I

Power quality – IEEE 519:

https://comsys.se/our-adf-technology/power-quality-ieee-519-2022/

PHASE-1

Phase 1: Data Preparation and Preprocessing

Step 1: Data Loading and Initial Exploration

~load the back-EMF data from the provided Excel files (`ideal.xlsx` and `nonideal.xlsx`) into our environment.

We will load the data into a pandas DataFrame

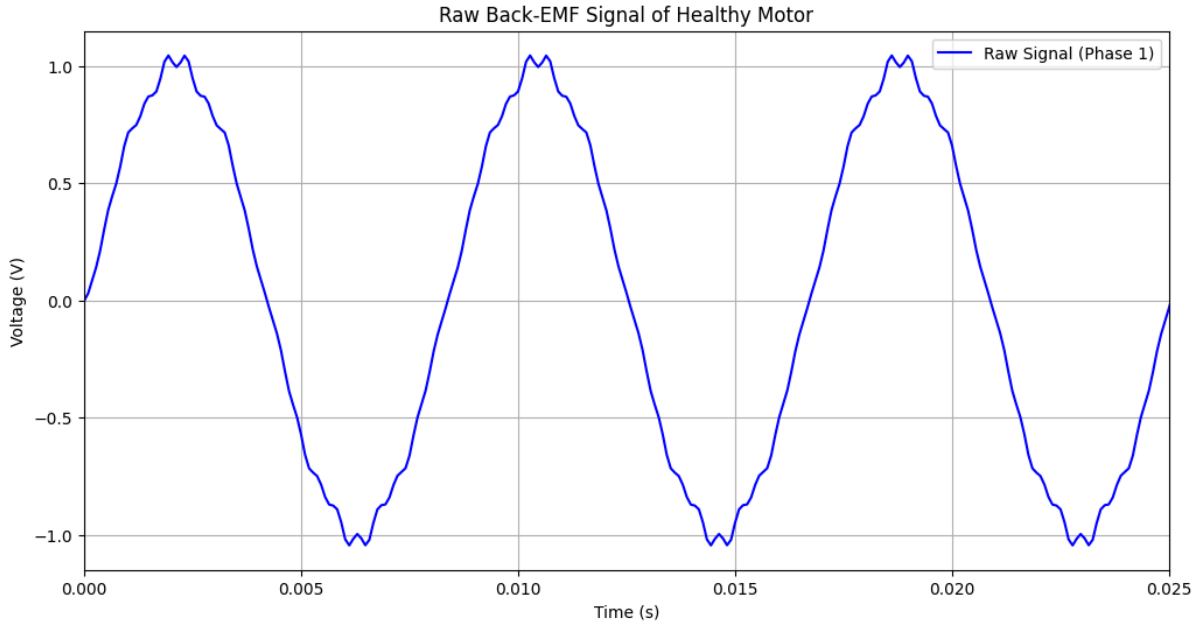
We will start with the "reference machine" to establish a baseline.

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import butter, filtfilt

# Load the healthy reference motor data from the 'ideal.xlsx' file
try:
    df_healthy = pd.read_excel("ideal.xlsx", sheet_name="reference machine")
    print("Successfully loaded 'reference machine' data.")
    print(df_healthy.head())
except FileNotFoundError:
    print("Error: 'ideal.xlsx' not found. Please ensure the file is in the
correct directory.")
```

```
Successfully loaded 'reference machine' data.
   Relative Time       Ph1       Ph2       Ph3
0       0.000000 -0.001326  0.000664  0.000662
1       0.000093  0.029115 -0.868701  0.839251
2       0.000185  0.085399 -0.874122  0.787423
3       0.000278  0.140933 -0.890885  0.747126
4       0.000370  0.212193 -0.946377  0.730963
```

Shows first 5 values from the Dataframe

```
# Extract the raw signal for Phase 1 to use as our example
time = df_healthy['Relative Time'].values
raw_signal = df_healthy['Ph1'].values

# Initial plot of the raw signal
plt.figure(figsize=(12, 6))
plt.plot(time, raw_signal, label='Raw Signal (Phase 1)', color='b')
plt.title('Raw Back-EMF Signal of Healthy Motor')
plt.xlabel('Time (s)')
plt.ylabel('Voltage (V)')
plt.xlim(0, 0.025) # Limit to a few cycles for clarity
plt.grid(True)
plt.legend()
plt.show()
```



This code loads the data and displays the first few rows, confirming we have the time and voltage data. The plot shows a clear sinusoidal waveform but with some visible high-frequency variations (noise).

Step 2: Data Cleansing (Low-Pass Filtering)

the raw signal contains noise 1. We will apply a low-pass digital filter to remove this noise while preserving the fundamental frequency (120 Hz)(will be derived later in this doc)* and its important fault-indicating harmonics

Concept:

We will use a **4th-order Butterworth filter.**

which is ideal because it has a maximally flat passband, ensuring it doesn't distort the amplitudes of the frequencies we want to keep. The cutoff is set to 2000 Hz to retain up to the 16th harmonic of the 120 Hz signal, which is more than enough for fault analysis

*Go through the following docs

Math behind Butterworth filters.pdf

Butterworth Filter in Machine Learning_.pdf

The above two docs are the notes on the research made by us.

```python
def apply_lowpass_filter(data, cutoff_freq=2000, sampling_rate=10800, order=4):
    """Applies a low-pass Butterworth filter with zero phase shift."""
    nyquist_freq = 0.5 * sampling_rate
    normal_cutoff = cutoff_freq / nyquist_freq
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    filtered_signal = filtfilt(b, a, data)
    return filtered_signal

# Apply the filter to our raw signal
filtered_signal = apply_lowpass_filter(raw_signal)

# Plot to compare raw vs. filtered signal
plt.figure(figsize=(12, 6))
plt.plot(time, raw_signal, label='Raw Signal', color='b', alpha=0.5)
plt.plot(time, filtered_signal, label='Filtered Signal',
color='r', linewidth=2)
plt.title('Comparison of Raw vs. Filtered Signal')
plt.xlabel('Time (s)')
plt.ylabel('Voltage (V)')
plt.xlim(0, 0.025)
plt.grid(True)
plt.legend()
plt.show()
```

⭐ (Click this) Common Questions that arise during the implementation.

```
import pandas as pd
from scipy.signal import find_peaks
import numpy as np

file_path = 'ideal.xlsx'
sheet_name = 'reference machine'

# Loading the data into a pandas DataFrame
try:
    df = pd.read_excel(file_path, sheet_name=sheet_name)
    print(f"Successfully loaded data from '{file_path}' sheet '{sheet_name}'.\n")
```

```
except FileNotFoundError:
    print(f"Error: The file '{file_path}' was not found. Please ensure it is available."
    exit()

# Extracting the 'Relative Time' and 'Ph1' voltage columns into NumPy arrays
time = df['Relative Time'].values
voltage_ph1 = df['Ph1'].values

# --- 3. Peak Finding ---
# Use scipy.signal.find_peaks to locate the indices of the peaks in the Ph1 signal.
# 'height=0.8' ensures we only find major peaks, ignoring small noise spikes.
# 'distance=50' sets a minimum separation of 50 data points between peaks,
# preventing multiple detections on the same wave crest.
peak_indices, _ = find_peaks(voltage_ph1, height=0.8, distance=50)

# --- 4. Results Display ---
# Use the found indices to get the corresponding time and voltage values for eac
peak_times = time[peak_indices]
peak_voltages = voltage_ph1[peak_indices]

print("--- Found Peaks in the Ph1 Waveform ---")
for i in range(min(2, len(peak_indices))):
    print(f"Peak {i+1}: Time = {peak_times[i]:.6f} seconds, Voltage = {peak_voltage

# --- 5. Verification ---
print("\n--- Verification of the Values ---")
if len(peak_times) >= 2:
    print(f"The first peak was found at Time = {peak_times[0]:.6f} s.")
    print(f"The second peak was found at Time = {peak_times[1]:.6f} s.")

    # Calculate the period and frequency as a bonus
    period = peak_times[1] - peak_times[0]
    frequency = 1 / period
    print(f"\nThe calculated period (time between peaks) is {period:.6f} s.")
    print(f"The corresponding fundamental frequency is {frequency:.2f} Hz.")
else:
```

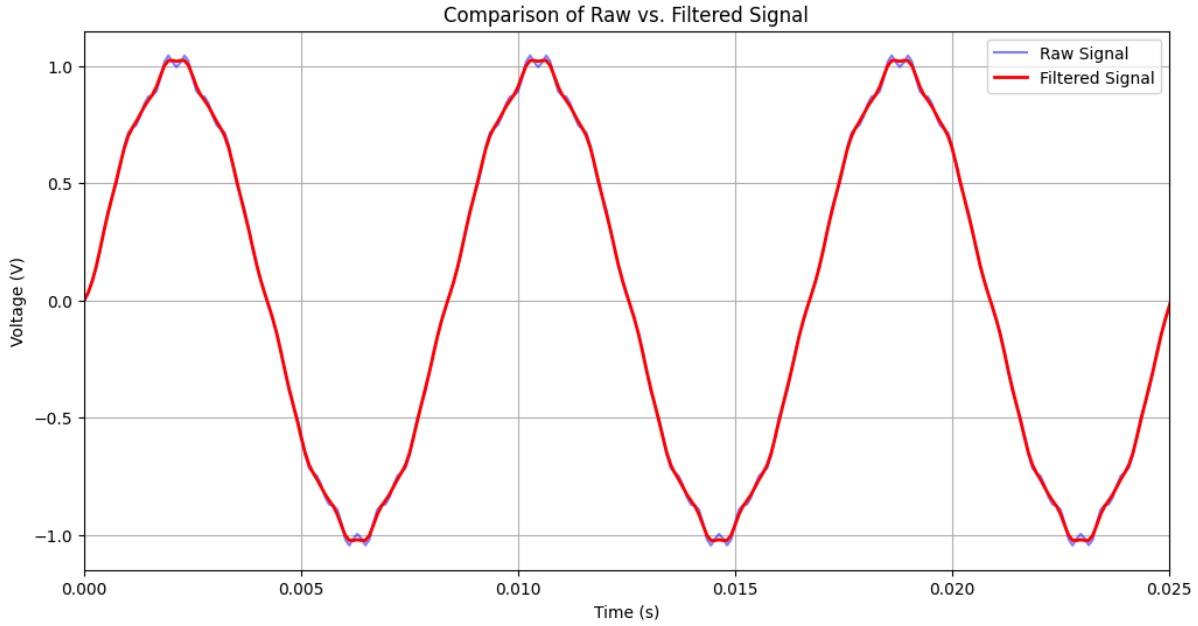
```
print("Could not find at least two peaks with the specified parameters.")
```

OUTPUT:

```
Successfully loaded data from 'ideal.xlsx' sheet 'reference machine'.

--- Found Peaks in the Ph1 Waveform ---
Peak 1: Time = 0.001944 seconds, Voltage = 1.0451 V
Peak 2: Time = 0.010278 seconds, Voltage = 1.0451 V

--- Verification of the Values ---
The first peak was found at Time = 0.001944 s.
The second peak was found at Time = 0.010278 s.

The calculated period (time between peaks) is 0.008333 s.
The corresponding fundamental frequency is 120.00 Hz.
```

```
def apply_lowpass_filter(data, cutoff_freq=2000, sampling_rate=10800, order=4):
    """Applies a low-pass Butterworth filter with zero phase shift."""
    nyquist_freq = 0.5 * sampling_rate
    normal_cutoff = cutoff_freq / nyquist_freq
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    filtered_signal = filtfilt(b, a, data)
    return filtered_signal

# Apply the filter to our raw signal
filtered_signal = apply_lowpass_filter(raw_signal)

# Plot to compare raw vs. filtered signal
plt.figure(figsize=(12, 6))
plt.plot(time, raw_signal, label='Raw Signal', color='b', alpha=0.5)
plt.plot(time, filtered_signal, label='Filtered Signal', color='r', linewidth=2)
plt.title('Comparison of Raw vs. Filtered Signal')
plt.xlabel('Time (s)')
plt.ylabel('Voltage (V)')
plt.xlim(0, 0.025)
plt.grid(True)
plt.legend()
plt.show()
```

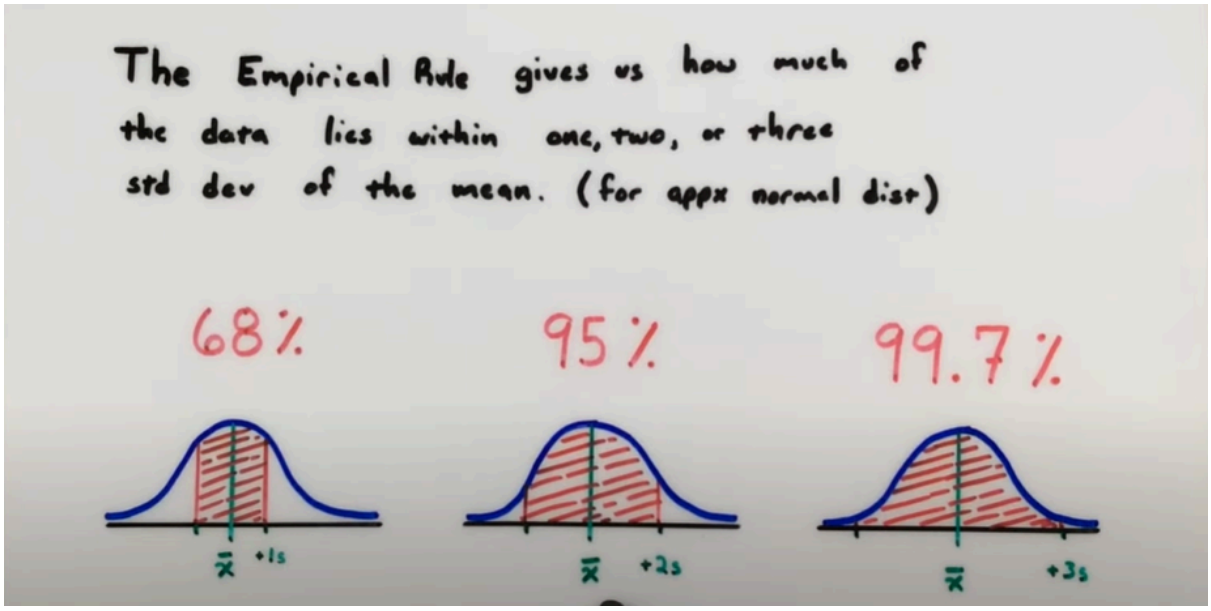
OUTPUT:



We can clearly observe that the filter had refined the signal curve.

Step 3: Outlier Removal

We'll use the **3-sigma rule** on a rolling basis to identify and correct them



For a normal distribution, 99.7% of data points lie within three standard deviations (σ) of the mean. Any point outside this range is a likely outlier. Instead of just removing it, which would disrupt the time series, we will replace it with an interpolated value from its neighbors.

Step 4: Normalization

The final preprocessing step is to normalize the voltage levels. This ensures that when we compare different motors, we are comparing the *shape* of their signals, not their absolute voltage scales, which might vary due to minor sensor differences.

Concept:

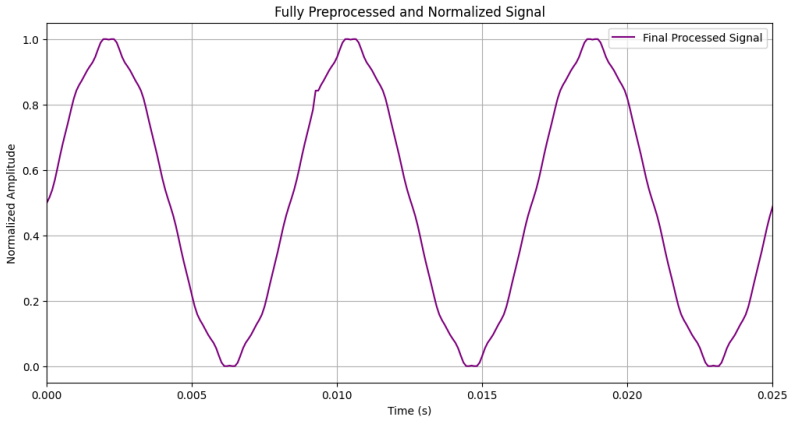
We will use **Min-Max scaling**

to transform the data into a range that lies between 0-1

This is a common technique that preserves the shape of the original data.

Implementation:

```python
def normalize_signal(signal):
    """Scales the signal to a [0, 1] range."""
    min_val = np.min(signal)
    max_val = np.max(signal)
    return (signal - min_val) / (max_val - min_val)

# Normalize our cleaned signal
normalized_signal = normalize_signal(signal_no_outliers)

# Plot the final processed signal
plt.figure(figsize=(12, 6))
plt.plot(time, normalized_signal, label='Final Processed
Signal', color='purple')
plt.title('Fully Preprocessed and Normalized Signal')
plt.xlabel('Time (s)')
plt.ylabel('Normalized Amplitude')
plt.xlim(0, 0.025)
plt.grid(True)
plt.legend()
plt.show()
```



The Y-axis now ranges from 0 to 1, indicating the signal is successfully normalized and ready for feature extraction.

Combining all the steps into a function.

```python
def preprocess_signal(raw_signal, sampling_rate=10800):

    # Step 1: Low-pass filtering
    filtered = apply_lowpass_filter(raw_signal, sampling_rate=sampling_rate)

    # Step 2: Outlier removal (on the filtered signal)
    cleaned = remove_outliers(filtered)

    # Step 3: Normalization
    normalized = normalize_signal(cleaned)

    return normalized

# --- Example Usage ---
# Load a new signal (e.g., from a faulty motor)
df_faulty = pd.read_excel("nonideal.xlsx", sheet_name="Type-1 variation C")
raw_faulty_signal = df_faulty['Ph1'].values

# Process it with one command
processed_faulty_signal = preprocess_signal(raw_faulty_signal)

print("Preprocessing complete. The data is now ready for Phase 2 and 3 analysis.")
```

```
Preprocessing complete. The data is now ready for Phase 2 and 3 analysis.
```

PHASE 2 Motor Fault Identification:

Assumptions Made During Analysis

- **Constant Sampling Rate:** Confirm and emphasize that a constant sampling rate of 10,800 Hz is assumed and verified across all datasets 23. This is critical for accurate frequency-domain analysis (Phase 3).
- **Clean Baseline Signal:** The "reference machine" data is considered the ideal healthy baseline. The preprocessing steps (filtering, outlier removal) aim to make all signals as "clean" as possible, assuming that any remaining deviations are due to faults and not measurement artifacts or external disturbances.
- **Electrical Angle Consistency:** The calculation of phase shift assumes a consistent relationship between the electrical angle and the fundamental frequency, which holds true for the steady-state conditions analyzed.

Crest factor ref: https://youtu.be/80dv_KDFc2E?feature=shared

Crest factor visualizer:

https://www.perplexity.ai/apps/1a341aa6-7ce9-42da-9c04-24bb74b02d2c

Objective: To extract quantitative features from the raw time-series back-EMF signals. We aim to capture the signal's energy, shape, and inter-phase relationships

These features complement each other:

Fault Type	Best Detected By
Bearing faults	Crest + Impulse Factor
Looseness	Impulse + Form Factor
Misalignment	Form Factor
Electrical faults	Form + Crest Factor

Section 1: Code Snippet - Imports and Configuration

```
python# Imports and Configuration Snippet
import pandas as pd
import numpy as np
from scipy import stats
from scipy.signal import correlate
import matplotlib.pyplot as plt
import seaborn as sns
import os

MOTOR_CONDITIONS = [
    # Conditions from ideal.xlsx
    ("ideal.xlsx", "reference machine"),
    ("ideal.xlsx", "Type-1 variation A"),
    ("ideal.xlsx", "Type-1 variation B"),
    ("ideal.xlsx", "Type-1 variation C"),
    ("ideal.xlsx", "Low temperature"),
    ("ideal.xlsx", "High temperature"),
    # Conditions from nonideal.xlsx
    ("nonideal.xlsx", "polarity issue"),
    ("nonideal.xlsx", "Core Fault 1"),
    ("nonideal.xlsx", "Core Fault 2"),
    ("nonideal.xlsx", "Type-2 variation A"),
```

```
    ("nonideal.xlsx", "Type-2 variation B")
]

SAMPLING_RATE = 10800
FUNDAMENTAL_FREQ = 120
```

Section 2: Code Snippet - Core Feature Functions

```
# ====================================================

def load_motor_data(file_path, sheet_name):
    """Loads motor data, returning a DataFrame or None on error."""
    if not os.path.exists(file_path):
        print(f"  → ERROR: The file '{file_path}' was not found. Please ensure it's in t
        return None
    try:
        df = pd.read_excel(file_path, sheet_name=sheet_name)
        return df[['Relative Time', 'Ph1', 'Ph2', 'Ph3']]
    except ValueError:
        print(f"  → ERROR: Worksheet '{sheet_name}' not found in '{file_path}'.")
        return None

def calculate_statistical_features(signal):
    """Calculates all time-domain statistical features for a single signal."""
    features = {
        'mean': np.mean(signal),
        'std_dev': np.std(signal),
        'rms': np.sqrt(np.mean(np.square(signal))),
        'peak': np.max(np.abs(signal)),
        'peak_to_peak': np.ptp(signal),
        'skewness': stats.skew(signal),
        'kurtosis': stats.kurtosis(signal),
```

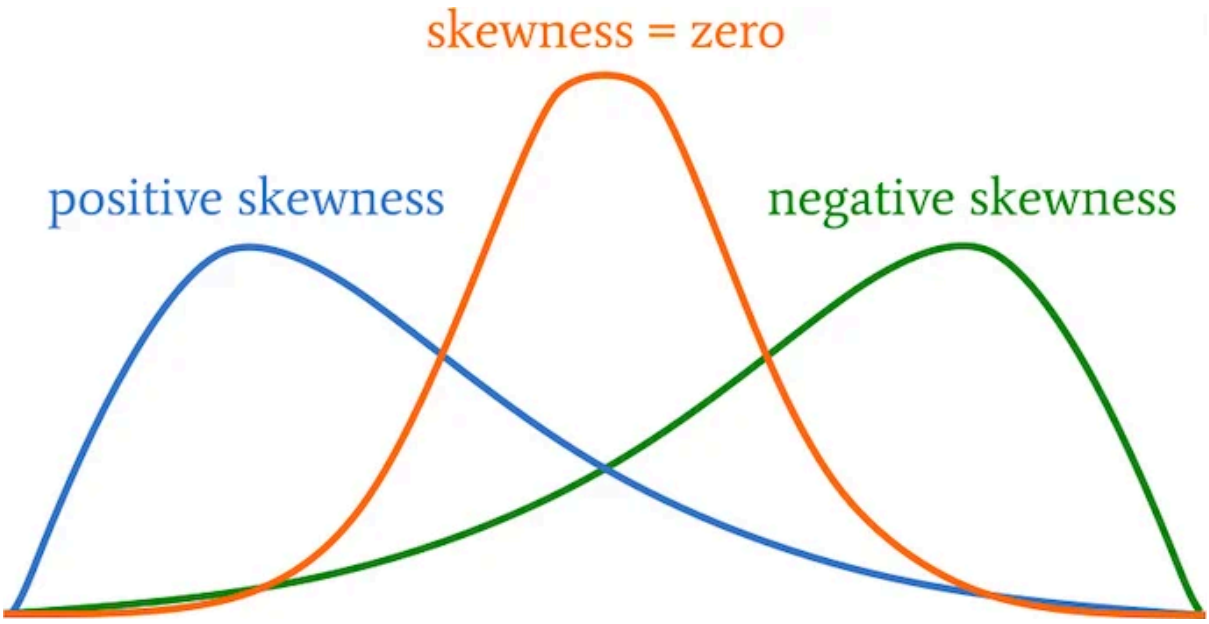
```
        'zero_crossings': len(np.where(np.diff(np.sign(signal)))[0])
    }
    return features

def calculate_shape_factors(signal):
    """Calculates waveform shape factors for a single signal."""
    rms_val = np.sqrt(np.mean(np.square(signal)))
    if rms_val == 0: return {'crest_factor': 0, 'form_factor': 0, 'impulse_factor': 0}

    mean_abs_val = np.mean(np.abs(signal))
    if mean_abs_val == 0: return {'crest_factor': 0, 'form_factor': 0, 'impulse_factor

    return {
        'crest_factor': np.max(np.abs(signal)) / rms_val,
        'form_factor': rms_val / mean_abs_val,
        'impulse_factor': np.max(np.abs(signal)) / mean_abs_val
    }

def calculate_phase_shift(sig1, sig2):
    """Calculates phase shift in degrees using cross-correlation."""
    correlation = correlate(sig1, sig2, mode='full')
    delay_samples = np.argmax(correlation) - (len(sig1) - 1)

    period_samples = SAMPLING_RATE / FUNDAMENTAL_FREQ
    phase_degrees = (delay_samples / period_samples) * 360
    return phase_degrees

def extract_all_time_features(df):
    """Master function to extract all time-domain features for a given motor's data
    features = {}

    # Per-Phase Feature Extraction
    for phase in ['Ph1', 'Ph2', 'Ph3']:
        signal = df[phase].values
        stats_features = calculate_statistical_features(signal)
        shape_features = calculate_shape_factors(signal)
```

```
        for key, val in {**stats_features, **shape_features}.items():
            features[f'{phase}_{key}'] = val

    # Inter-Phase Feature Extraction
    ph1, ph2, ph3 = df['Ph1'].values, df['Ph2'].values, df['Ph3'].values

    features['phase_shift_12'] = calculate_phase_shift(ph1, ph2)
    features['rms_balance_12'] = features['Ph1_rms'] / features['Ph2_rms'] if featu

    return features

def normalize_features(feature_df):
    """Normalizes all feature columns in the DataFrame using Min-Max scaling."""
    features_to_normalize = feature_df.drop(columns=['condition'])
    normalized_features = (features_to_normalize - features_to_normalize.min()) /
                          (features_to_normalize.max() - features_to_normalize.min())

    return pd.concat([feature_df[['condition']], normalized_features], axis=1)
```

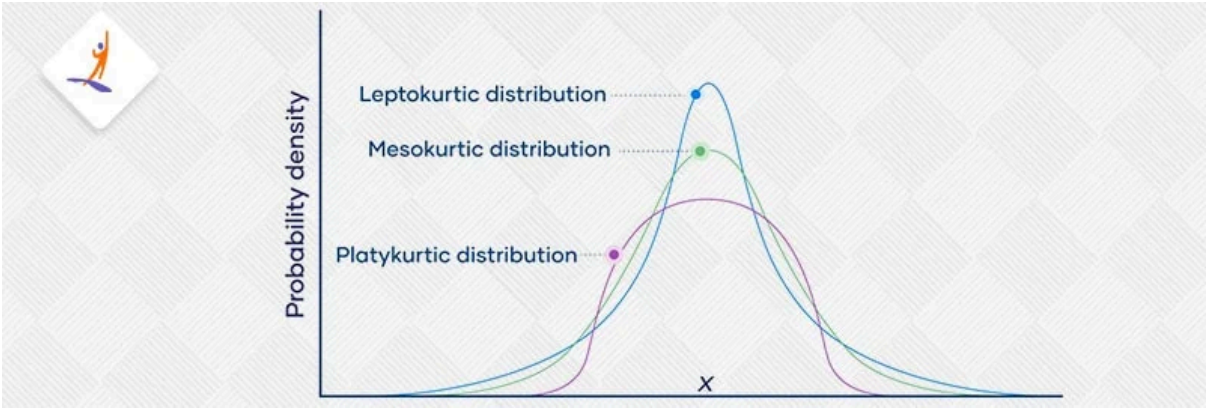
`load_motor_data`: This function safely loads data from an Excel file. checks if the file exists and if the sheet name is correct, preventing the program from crashing

`calculate_statistical_features`:

- **RMS (Root Mean Square):** Measures the signal's effective power or energy level.
- **Peak-to-Peak:** Measures the full range of the voltage swing.
- **Skewness:** Measures the signal's asymmetry. A value of 0 is perfectly symmetric.
-



- **Kurtosis:** Measures the "tailedness" or "peakedness" of the signal compared to a normal distribution.



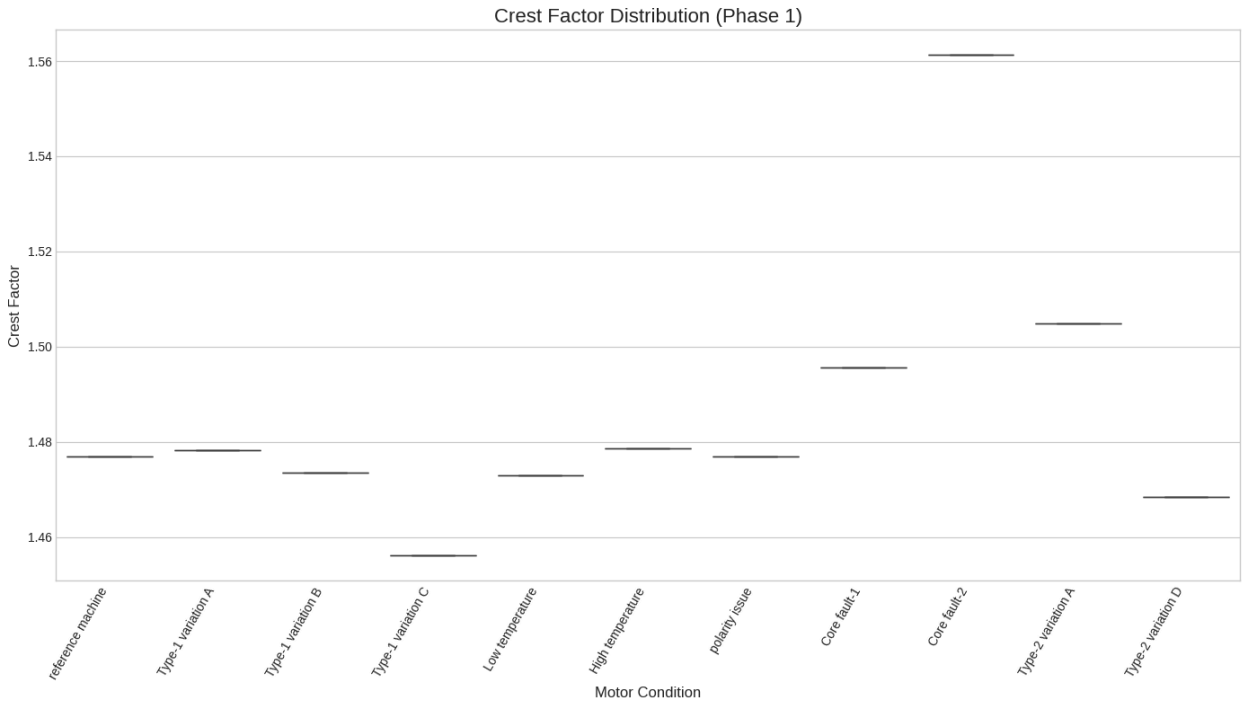
`calculate_shape_factors`**:** These functions calculate dimensionless ratios that describe the *shape* of the waveform

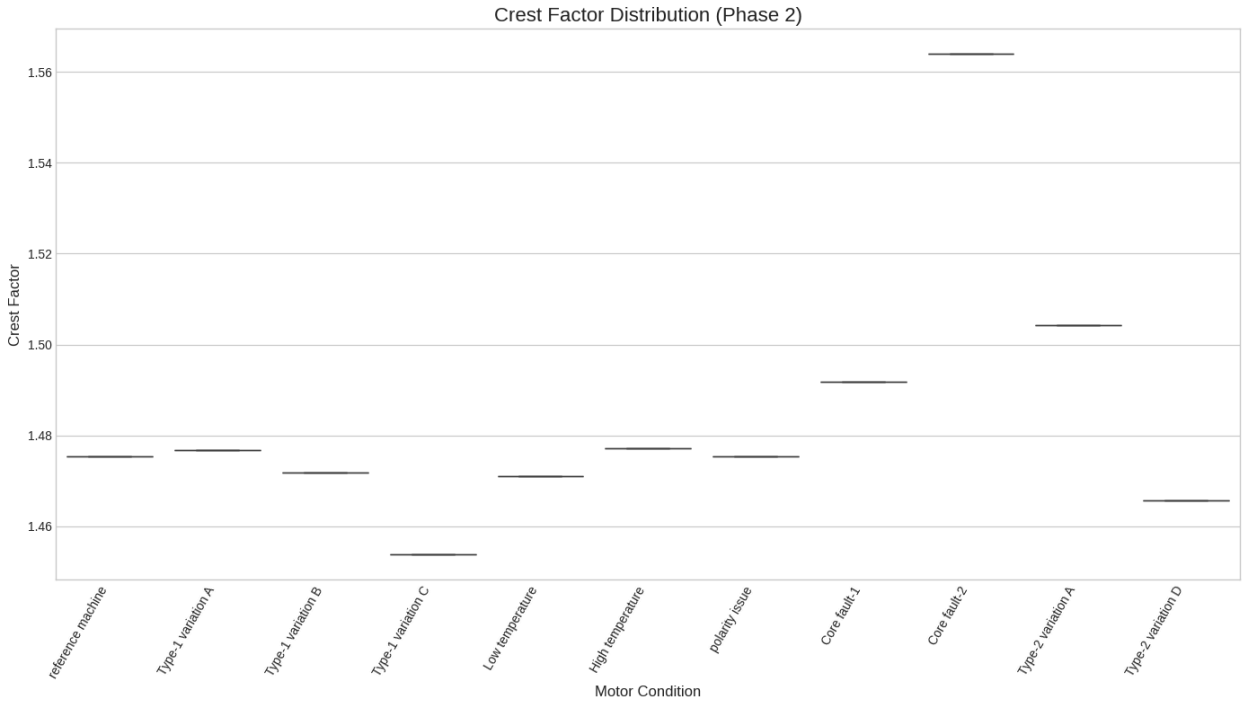
-

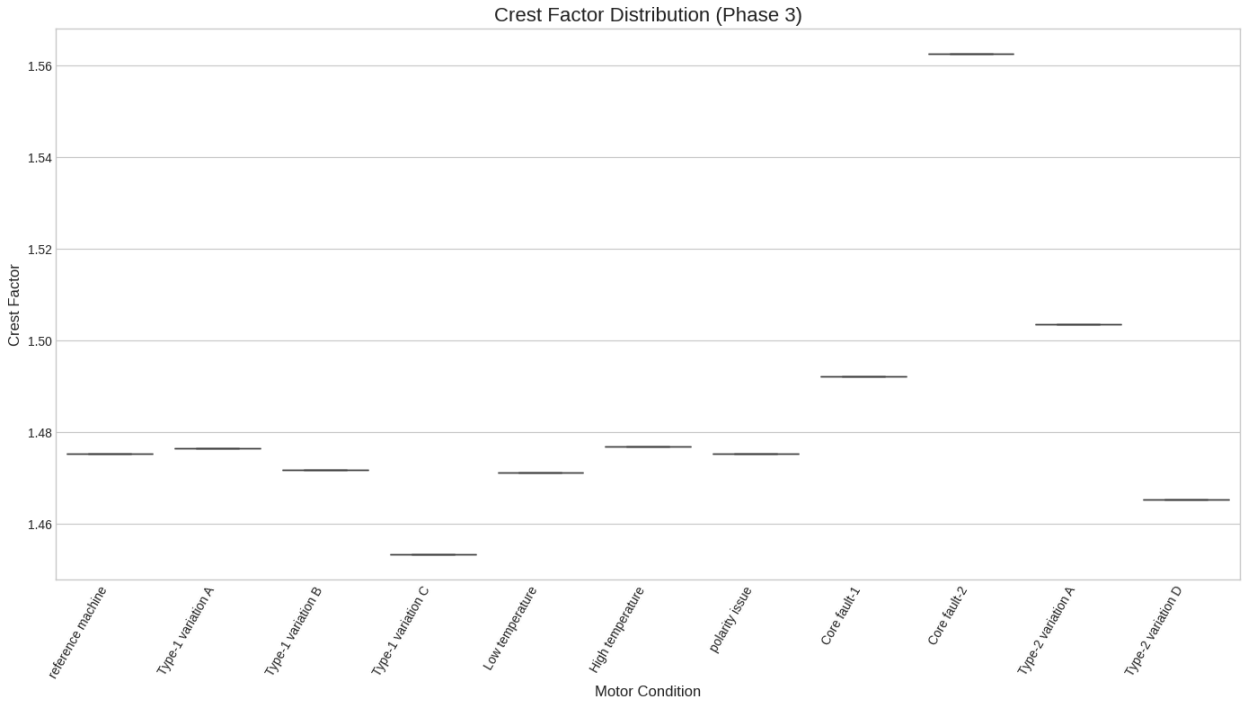
Crest Factor: The key feature for detecting sharp impacts or "spikes" in the signal.

https://youtu.be/80dv_KDFc2E?feature=shared

- **What it Measures:** The "spikiness" of the signal. A perfect sine wave has a Crest Factor of 1.414.
- **How to Interpret the Graph:** Look for conditions where the boxplot is significantly higher than the "reference machine."
- **What it Signifies for Faults:**
    - **High Crest Factor:** This is a strong indicator of **mechanical impacting**, such as from bearing wear, gear tooth damage, or physical rubbing. The "Core Fault" conditions will likely show a higher Crest Factor because the damaged core can cause irregular magnetic forces that create sharp peaks in the back-EMF.
    - **Low Crest Factor:** This can indicate a signal that is "flattened" or "clipped" at the top, a form of distortion seen in some electrical faults.




Crest Factor Distribution (Phase 2)
Crest Factor
1.56
1.54
1.52
1.50
1.48
1.46
reference machine
Type-1 variation A
Type-1 variation B
Type-1 variation C
Low temperature
High temperature
polarity issue
Core fault-1
Core fault-2
Type-2 variation A
Type-2 variation D
Motor Condition


Crest Factor Distribution (Phase 3)
Crest Factor
1.56
1.54
1.52
1.50
1.48
1.46
reference machine
Type-1 variation A
Type-1 variation B
Type-1 variation C
Low temperature
High temperature
polarity issue
Core fault-1
Core fault-2
Type-2 variation A
Type-2 variation D
Motor Condition

```
def calculate_phase_shift(sig1, sig2):
    """Calculates phase shift in degrees using cross-correlation."""
    correlation = correlate(sig1, sig2, mode='full')
    delay_samples = np.argmax(correlation) - (len(sig1) - 1)

    period_samples = SAMPLING_RATE / FUNDAMENTAL_FREQ
    phase_degrees = (delay_samples / period_samples) * 360
    return phase_degrees
```

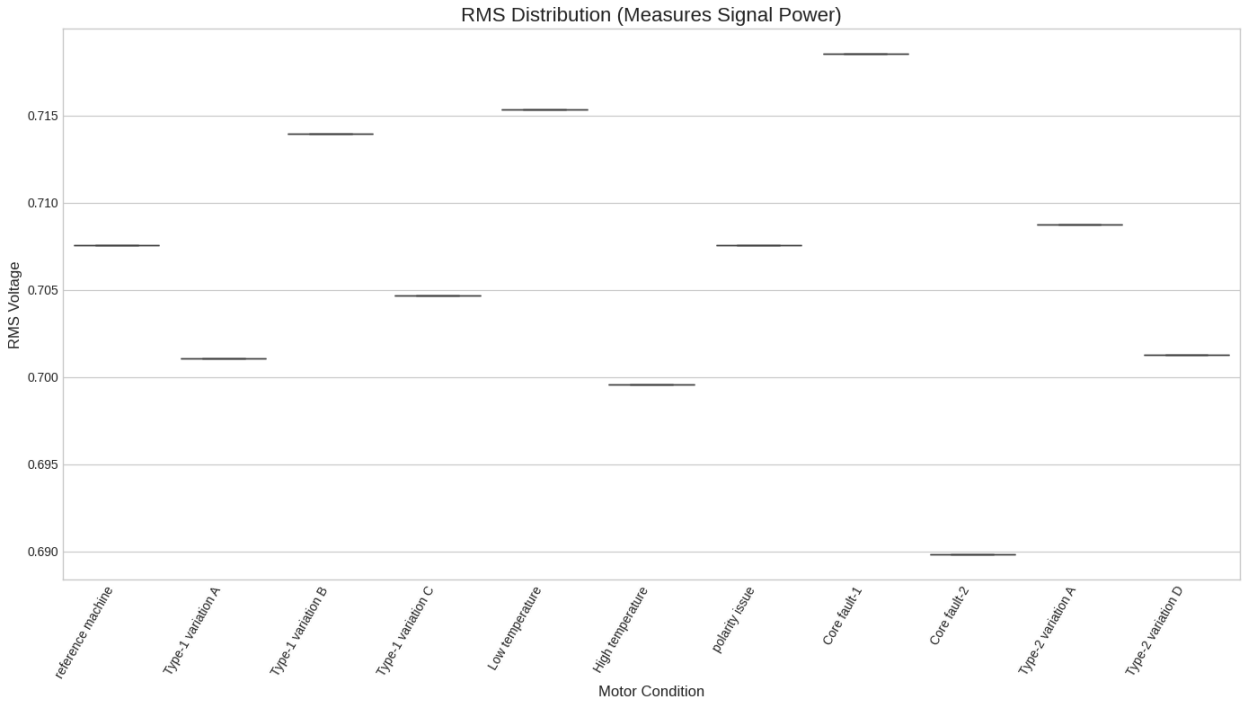
⭐ (click this)Concept of Phase Shift in a 3-Phase Motor

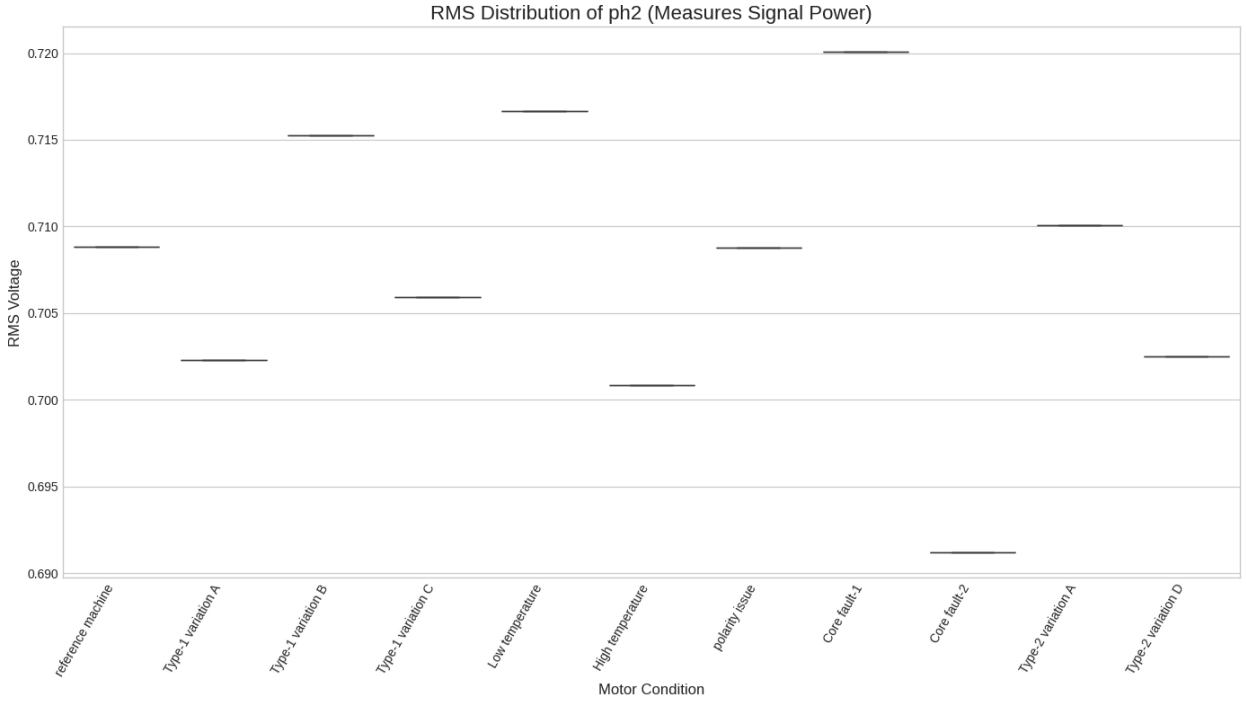
💡 Section 3: Graph Significance - Interpreting the Results

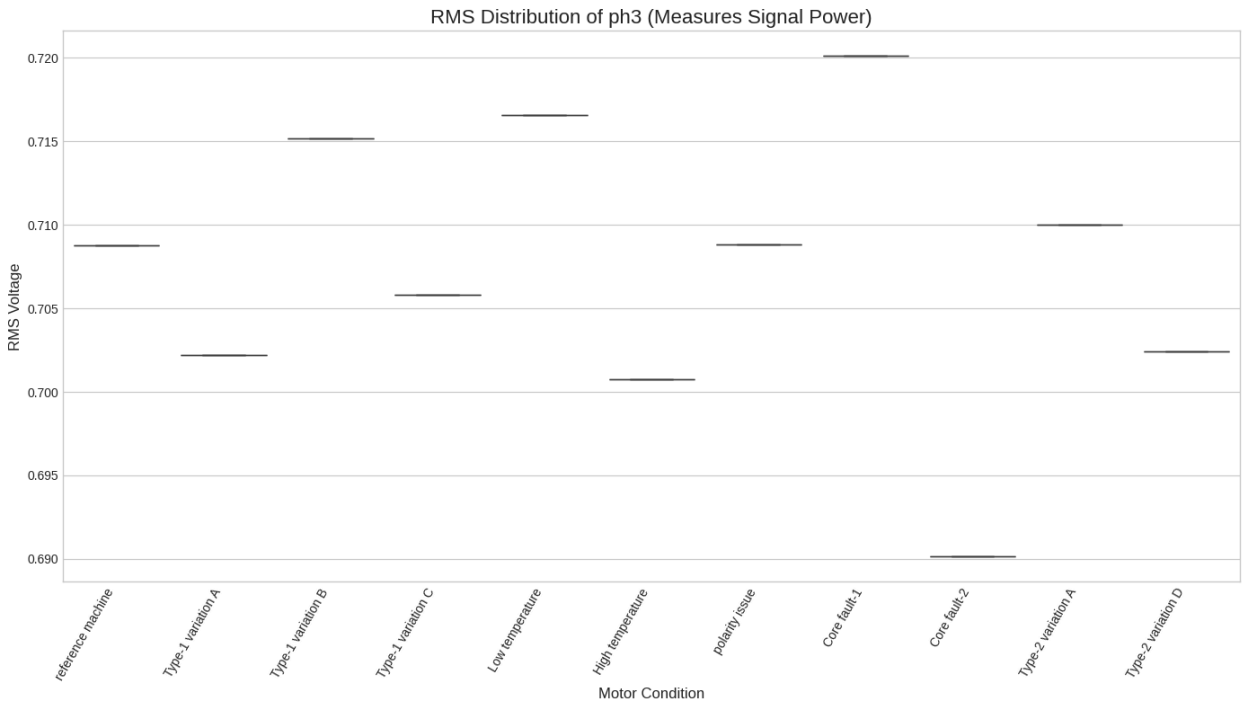
understanding what the analysis tells us about motor health. The plots generated by the universal plotting function provide a "diagnostic dashboard."

Graph 1: RMS (Root Mean Square) Distribution

- **What it Measures:** The overall energy or power of the signal.
- **How to Interpret the Graph:** Look for conditions with a median RMS value that is noticeably different from the healthy reference motor.
- **What it Signifies for Faults:**
    - **Low RMS:** This indicates a loss of power. You should see a significantly lower RMS for the **"High temperature"** condition, as extreme heat increases winding resistance and reduces the motor's efficiency. **"Core Fault 2,"** being a severe fault, will also likely show a lower RMS due to energy losses in the damaged core.
    - **High RMS:** This is less common but could indicate a fault that is adding unwanted energy to the system, possibly at harmonic frequencies.

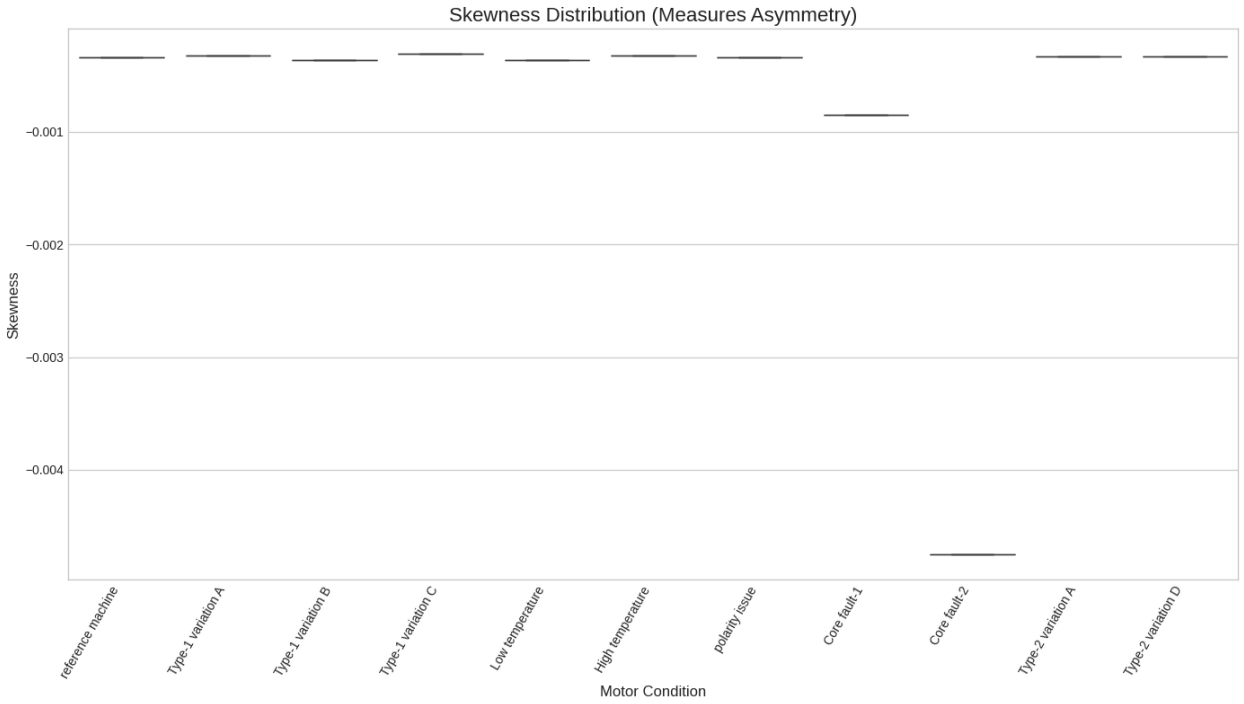

RMS Distribution (Measures Signal Power)
RMS Voltage
0.715
0.710
0.705
0.700
0.695
0.690
reference machine
Type-1 variation A
Type-1 variation B
Type-1 variation C
Low temperature
High temperature
polarity issue
Core fault-1
Core fault-2
Type-2 variation A
Type-2 variation D
Motor Condition

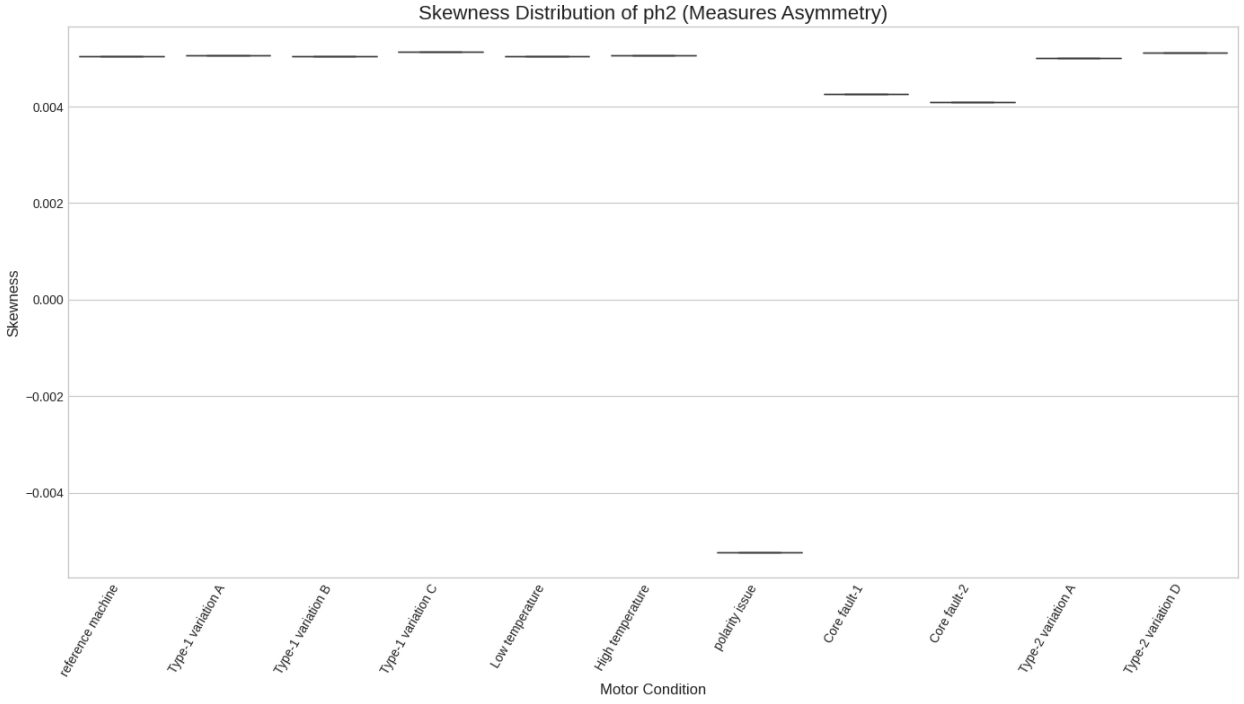

RMS Distribution of ph2 (Measures Signal Power)
RMS Voltage
0.720
0.715
0.710
0.705
0.700
0.695
0.690
reference machine
Type-1 variation A
Type-1 variation B
Type-1 variation C
Low temperature
High temperature
polarity issue
Core fault-1
Core fault-2
Type-2 variation A
Type-2 variation D
Motor Condition

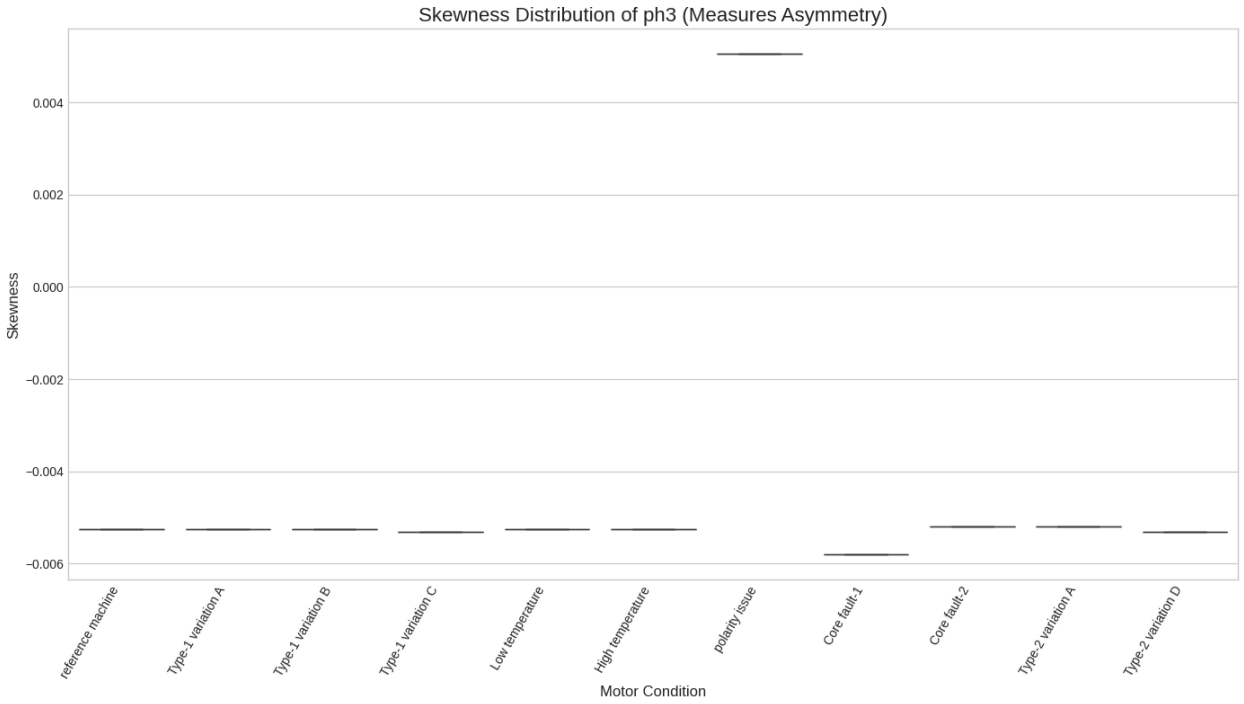


Graph 2: Skewness Distribution

- **What it Measures:** The symmetry of the waveform. A healthy AC signal should be perfectly symmetrical around zero, giving a skewness value very close to 0.
- **How to Interpret the Graph:** Look for any condition where the boxplot is not centered on zero.
- **What it Signifies for Faults:**
    - **Non-Zero Skewness:** This points to an electrical problem. It means the positive half of the wave has a different shape than the negative half. This can be caused by issues in the motor windings or problems with the power supply that introduce a DC offset. The "Type-1" and "Type-2" variations might show some skewness.

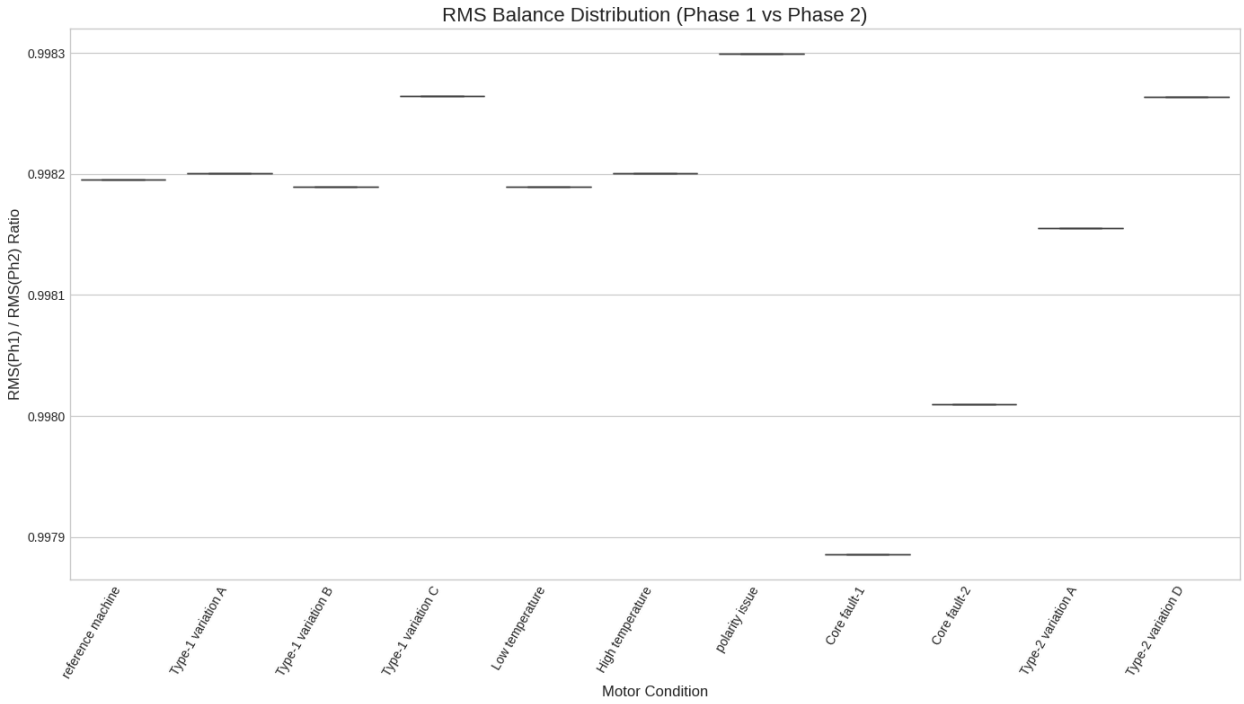

Skewness Distribution (Measures Asymmetry)
Skewness
−0.001
−0.002
−0.003
−0.004
reference machine
Type-1 variation A
Type-1 variation B
Type-1 variation C
Low temperature
High temperature
polarity issue
Core fault-1
Core fault-2
Type-2 variation A
Type-2 variation D
Motor Condition


Skewness Distribution of ph2 (Measures Asymmetry)
Skewness
0.004
0.002
0.000
−0.002
−0.004
reference machine
Type-1 variation A
Type-1 variation B
Type-1 variation C
Low temperature
High temperature
polarity issue
Core fault-1
Core fault-2
Type-2 variation A
Type-2 variation D
Motor Condition



Graph 3: RMS Balance Distribution

- **What it Measures:** The ratio of power between Phase 1 and Phase 2 (`RMS(Ph1) / RMS(Ph2)`). For a healthy, balanced three-phase motor, this ratio should be almost exactly **1.0**.
- **How to Interpret the Graph:** Look for any condition that is **not** tightly centered at 1.0.
- **What it Signifies for Faults:**
    - **Ratio ≠ 1.0:** This is the "smoking gun" for **phase imbalance**. This graph will be the most powerful tool for identifying the **"polarity issue."** When one phase is wired backwards, its relationship to the others is fundamentally broken, causing a severe imbalance in the phase voltages. This plot should show the "polarity issue" as a dramatic outlier, while all other conditions (even severe faults like Core Fault 2) will likely remain close to a balanced ratio of 1.0. This perfectly demonstrates why specific features are excellent for diagnosing specific faults.



Phase 3: Frequency-Domain Analysis

In electricity, the "heartbeat" is the standard AC power, and "irregular beats" are called harmonics.

Harmonics ref: https://youtu.be/EYRmB1aNh9I

Why do we care about the 5th, 7th, and 11th in motors?
These specific harmonics are often produced by modern electronic equipment, especially devices that convert AC to DC and then back to AC (like variable frequency drives, or VFDs, which control motor speed). When these harmonics flow into motors, they don't help the motor run; instead, they cause problems:

- **Increased Heat:** Harmonics cause extra currents to flow in the motor windings, leading to more heat than usual. This heat can degrade insulation, shorten the motor's lifespan, and even lead to premature failure.
- **Torque Pulsations:** Harmonics can cause the motor's rotating force (torque) to pulsate or become uneven. This can lead to vibrations, noise, and mechanical stress on the motor and connected equipment.

- **Reduced Efficiency:** The energy associated with harmonics is wasted as heat, meaning the motor uses more power to do the same amount of work, increasing electricity bills.
- **Resonance:** In some cases, harmonics can interact with the motor's inherent electrical properties, leading to resonance. This amplifies the harmonic currents and voltages to dangerously high levels, causing severe damage.

General Fault Patterns Associated with Harmonics:

- **Overheating:** Motors feel unusually hot to the touch.
- **Increased Noise/Vibration:** The motor hums louder or vibrates more than usual.
- **Premature Motor Failure:** Motors failing more frequently than expected.
- **Nuisance Tripping:** Circuit breakers or protective devices tripping without an apparent overload.
- **Capacitor Bank Damage:** Power factor correction capacitors failing prematurely due to overheating from harmonic currents.

Total Harmonic Distortion.

It's a single number that tells us how "distorted" an electrical waveform (voltage or current) is, compared to its ideal, pure shape. Think of it as a quality score for your electricity. A lower THD is better, indicating cleaner power.

$$\text{THD} = \frac{\sqrt{V_2^2 + V_3^2 + \cdots + V_n^2}}{V_1}$$

- **V1 (Denominator):** This is the **RMS (Root Mean Square) voltage of the fundamental frequency**.

- **What it means:** This is the *good, usable voltage* at your standard frequency (e.g., 230V at 50 Hz). It's the voltage that actually does the work.

- **V22+V32+...+Vn2 (Numerator):** This is the **RMS sum of *all* the harmonic voltages** (excluding the fundamental).

  - **V2,V3,...,Vn:** These represent the RMS voltages of the individual harmonic components (2nd, 3rd, 4th, 5th, and so on, up to the 'n'th harmonic you're measuring). Each of these is an "unwanted ripple."
  - **What the whole numerator means:** It's a way to combine the "strength" of all the unwanted harmonic ripples into a single value. We square each individual harmonic voltage, add them up, and then take the square root to get an overall "harmonic voltage."

You divide the total "strength" of the harmonics by the strength of the good, fundamental voltage. This gives you a ratio, which is then usually multiplied by 100 to express it as a
percentage.

Example:

If your fundamental voltage is 230V, and the combined "harmonic voltage" (the numerator) is 23V, then:
THD = (23V / 230V) = 0.1
THD% = 0.1 * 100 = 10%
This means 10% of your total voltage is made up of unwanted harmonics.

It's a ratio: (Total Unwanted Harmonic Voltage) / (Good Fundamental Voltage).

```
==================================================
    Displaying Key Frequency Feature Values
==================================================
            condition  Ph1_fundamental_amplitude  Ph1_harmonic_3_amp  Ph1_harmonic_5_amp  Ph1_thd
0   reference machine                     0.9991              0.0045              0.0032   3.1519
1  Type-1 variation A                     0.9898              0.0044              0.0042   3.2153
2  Type-1 variation B                     1.0081              0.0045              0.0023   3.0844
3  Type-1 variation C                     0.9954              0.0044              0.0028   2.6578
4     Low temperature                     1.0101              0.0045              0.0020   3.0715
5    High temperature                     0.9877              0.0044              0.0044   3.2321
6      polarity issue                     0.9991              0.0045              0.0032   3.1519
7        Core fault-1                     1.0142              0.0049              0.0021   3.0776
8        Core fault-2                     0.9670              0.0010              0.0921   9.8718
9  Type-2 variation A                     1.0003              0.0045              0.0034   3.3345
```

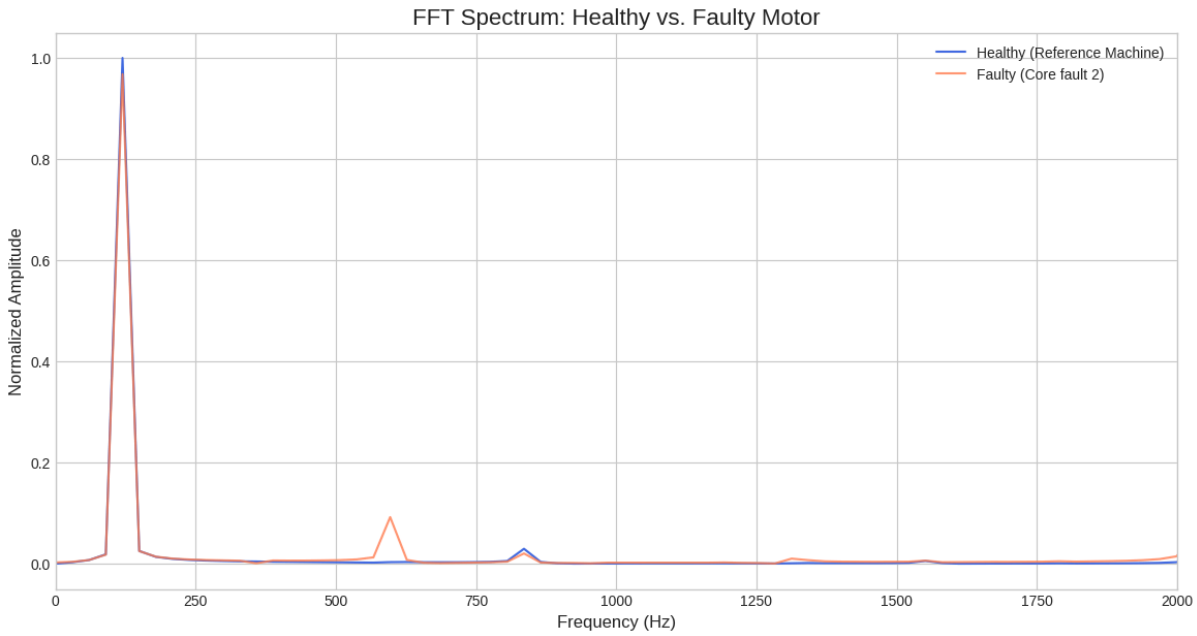
It loops through all 11 motor conditions, extracts the frequency features for each, saves them to a new CSV file, and then prints a summary table.

Visualization and Project Notes

This section is for creating the diagnostic plots and providing the explanations for your project notes.

FFT Spectrum Comparison: It plots the frequency spectrum of a healthy motor against a severely faulty one ("Core Fault 2"). This visually demonstrates how faults introduce harmonic distortions.

THD Distribution: It creates a boxplot comparing the Total Harmonic Distortion (THD) across all 11 motor conditions. This plot serves as a powerful single-glance diagnostic tool.



Graph 1: FFT Spectrum Comparison What it Shows: This plot overlays the frequency content of a healthy motor with a faulty one. The Y-axis is the strength (amplitude) of each frequency component, and the X-axis is the frequency in Hz.

What it Signifies:

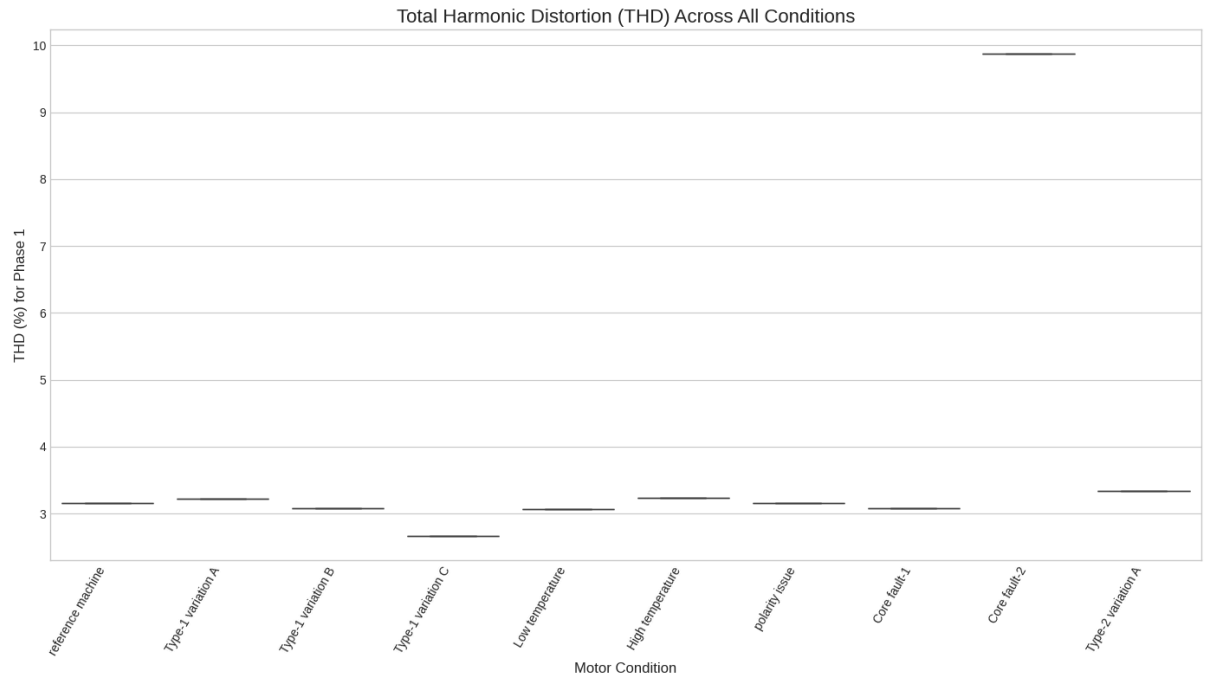
Healthy Motor (Blue Line): Notice the single, large, clean spike at 120 Hz. This is the fundamental frequency, where almost all the signal's energy is concentrated. The rest of the spectrum has a very low "noise floor."

Faulty Motor (Orange Line):

Harmonic Peaks: You can now clearly see new, significant peaks appearing at integer multiples of 120 Hz (e.g., 360 Hz, 600 Hz, 840 Hz). These are the harmonics created by the fault-induced waveform distortion.

Increased Noise Floor: The entire baseline of the faulty motor's spectrum is higher, indicating more broadband noise and energy spread across all frequencies.

Conclusion: This plot proves that the FFT is a powerful tool for making hidden distortions visible and quantifiable .



Graph 2: THD Distribution What it Shows: This boxplot provides a summary of the Total Harmonic Distortion (THD) for each motor condition. THD is a single number that represents the total amount of distortion in the signal .

What it Signifies:

Low THD: The "reference machine" and other healthy or mildly faulty conditions will have a very low THD, indicating their waveform is very close to a pure sine wave.

High THD: Conditions like "Core Fault 1", "Core Fault 2", and potentially some "Type-2" variations will show a significantly higher THD. This is a direct measure of how much their waveforms deviate from the ideal.

Diagnostic Power: This graph is arguably the most important output of Phase 3. It can act as a high-level "health check." A simple rule like "if THD > X%, flag motor for inspection" could be a very effective first-pass diagnostic test . You can clearly rank the faults by severity based on their THD levels.

PHASE-4: Machine Learning & Feature Extraction

4.1 Goals of Phase-4

- Merge all time-domain and frequency-domain features into a single “master” dataset.
- Standardise, encode, and visualise the full feature space.
- Train an illustrative classifier (SVM) despite the single-sample-per-class constraint.
- Evaluate with leave-one-out cross-validation (LOOCV) and document limitations.
- Produce clear plots: PCA scatter, confusion matrix, and feature-importance bar chart.

Column groups	Examples
Identification	`condition`
Time-domain	`Ph1_rms` , `Ph1_crest_factor` , `phase_shift_12` , …
Frequency-domain	`Ph1_thd` , `Ph1_harmonic_5_amplitude` , …

Total feature count: **95** (34 time + 61 frequency)

```
ythonfrom sklearn.preprocessing import LabelEncoder, StandardScaler

y_raw = features['condition']
X_raw = features.drop('condition', axis=1)

le = LabelEncoder()
y  = le.fit_transform(y_raw)
 # categorical → integer

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_raw)
 # zero-mean, unit-variance
```

> Note: each motor condition currently has one row → no stratified train/test split possible.LOOCV is adopted purely

> to illustrate a workflow, not to claim statistical validity.