Industrial Data Analytics (ME 217) Mini Project

CWRU Bearing : Team 3.1

Rucha Jatin Prabhu- 230003060
Koyna Pandit- 230003034
Sri Varsha- 230003074

PROBLEM STATEMENT

To analyze ball bearing test data to detect faults in bearings. This test data is vital for early detection and diagnosis in rotating machinery.

Context

Tests were done with a HP motor to collect acceleration data at different loads and speeds, close to and away from the motor bearings. The goal is to identify if the bearings are healthy or have faults like ball, inner, or outer issues.

Purpose

Early fault detection in bearings using machine learning. The goal is to predict bearing conditions accurately, enhancing equipment reliability and reducing maintenance costs.

Outcome Expectation

Accurately classify faults with a weighted F1-score exceeding 0.9, minimizing misclassifications like labeling faults as "Healthy". Using advanced techniques to pick out the most useful information helps provide clear insights. This supports predictive maintenance and makes operations more reliable.

Data Collection

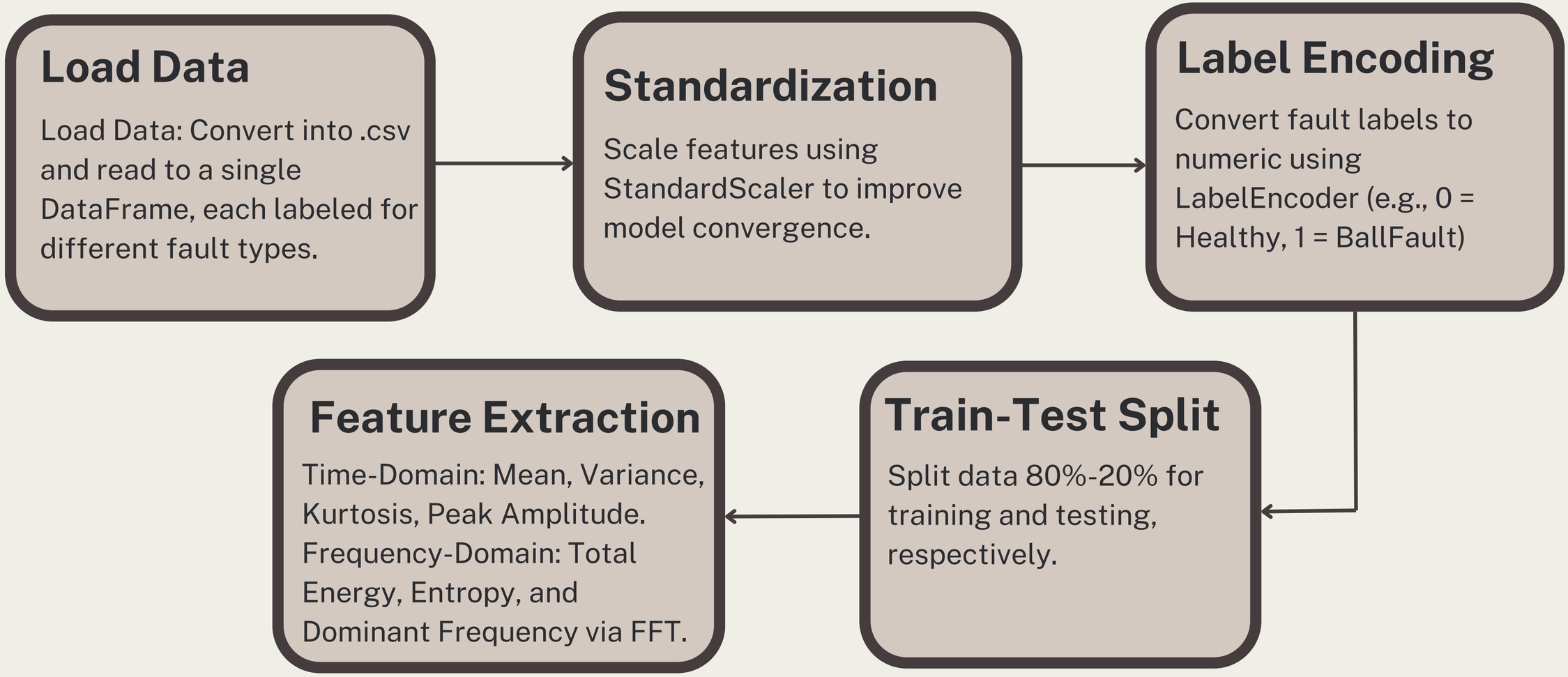
Data Source: Acceleration data for normal and faulty bearings, with faults induced by electro-discharge machining.

Data Files: Multiple .xlsx files from sensors at different positions.

Dataset: 12,000 samples with time and frequency domain features indicating bearing health.

DATA PREPROCESSING

Includes the specific techniques used to handle the dataset, clean and format it, and prepare it for machine learning.



FEATURE EXTRACTION

Time-Domain Features:

- Mean
- Variance
- Kurtosis: Tells us how much the data has outliers or how "peaked" the data distribution is.
- Peak Amplitude

Frequency-Domain Features:

- FFT (Fast Fourier Transform)
- Total Energy
- Signal Entropy: Shows how random or unpredictable a signal is.
- Dominant Frequency

Feature Selection:

- Top 7 features selected using ANOVA F-value analysis after feature scaling, ensuring the most statistically relevant features are retained for fault classification.

MODEL TRAINING

Algorithm Used

- Random Forest was chosen for its robustness in handling complex datasets, ability to rank feature importance for interpretability, and reliable performance across diverse fault classification tasks.

80-20 Data Split

Hyperparameter Tuning

- Utilized RandomizedSearchCV to optimize hyperparameters.
- Tuned parameters include:
  1. **n_estimators:** Number of trees.
  2. **max_depth**: Maximum depth of trees.
  3. **min_samples_split**: Minimum samples for a split.
- Resulted in optimal parameters

Final Model

- A weighted F1-score of 0.92, indicating high accuracy and balanced classification performance.
- Significant reduction in faults misclassified as “Healthy,” addressing the critical issue.

MODEL EVALUATION

Test Set Split:
Reserved 20% of data for evaluating the final model performance.

Classification Report:
Precision, Recall, and F1-score provided for each class (BallFault, Ballfault, Healthy, Innerfault). Achieved an overall weighted F1-score of 0.92, indicating strong predictive performance across classes.

Confusion Matrix:
Visualized true vs. predicted labels using a heatmap. Helps identify common misclassifications and assess class-wise accuracy.

Key Metrics:
High accuracy and F1-scores indicate effective classification. Results suggest the model handles complex patterns in bearing fault detection well.

KEY CODE SNIPPETS

1. Label Encoding: Encodes categorical labels (y) into integers using **LabelEncoder()**. The mapping of classes is displayed.

```
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
label_mapping = {label: idx for idx, label in enumerate(label_encoder.classes_)}
print("Label Encoding Mapping:", label_mapping)
```

```
Label Encoding Mapping: {'BallFault': 0, 'Ballfault': 1, 'Healthy': 2, 'Innerfault': 3}
```
(Output)

2. Feature Extraction

- FFT, Energy, and Entropy: Extracts advanced frequency-domain features (total_energy, signal_entropy, dominant_frequency) from signals to enhance model input.

```
def extract_frequency_features(signal):
    fft_values = np.abs(fft(signal))
    total_energy = np.sum(fft_values ** 2)
    signal_entropy = entropy(fft_values)
    dominant_frequency = np.argmax(fft_values)
    return total_energy, signal_entropy, dominant_frequency
```

KEY CODE SNIPPETS

- Add Extracted Features: Combines extracted frequency features with the existing dataset.

```
frequency_features = np.array([extract_frequency_features(row) for row in X.values])
freq_df = pd.DataFrame(frequency_features, columns=['total_energy', 'signal_entropy', 'dominant_frequency'])
X = pd.concat([X, freq_df], axis=1)
```

3. Feature Scaling and Selection

- Standardization: Scales features to have zero mean and unit variance for optimal model performance.

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

- SelectKBest: Selects the k most relevant features based on their statistical significance (f_classif).

```
k_best = min(7, X.shape[1])  # Adjust k dynamically
selector = SelectKBest(score_func=f_classif, k=k_best)
X_selected = selector.fit_transform(X_scaled, y_encoded)
selected_features = X.columns[selector.get_support(indices=True)]
print("Selected Features:", selected_features)
```

KEY CODE SNIPPETS

4. Model Training and Hyperparameter Tuning

- Define Hyperparameter Grid: Creates a grid with possible parameter values for tuning a Random Forest.

```
param_grid = {
    'n_estimators': [100, 150],
    'max_depth': [5, 10],
    'min_samples_split': [2, 5]
}
```

- Randomized Search: Uses **RandomizedSearchCV** to efficiently explore the hyperparameter space and find the best Random Forest configuration.

```
random_search = RandomizedSearchCV(
    RandomForestClassifier(random_state=42), param_distributions=param_grid, scoring='f1_weighted',
    cv=5, n_iter=5, n_jobs=-1, random_state=42, verbose=1
)
random_search.fit(X_selected, y_encoded)
best_rf_clf = random_search.best_estimator_
```

KEY CODE SNIPPETS

5. Model Evaluation

- Train-Test Split: Splits the dataset into training and testing sets for evaluation.

```
X_train, X_test, y_train, y_test = train_test_split(X_selected, y_encoded, test_size=0.2, random_state=42)
```

- Train and Predict: Fits the best Random Forest model and makes predictions.

```
best_rf_clf.fit(X_train, y_train)
y_pred = best_rf_clf.predict(X_test)
```

- Classification Metrics: Prints a classification report, calculates F1-score, and visualizes a confusion matrix.

```
print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=label_encoder.classes_))
```

```
f1 = f1_score(y_test, y_pred, average='weighted')
print("F1 Score:", f1)
```

(Output)

```
Classification Report:
              precision    recall  f1-score   support

   BallFault       0.84      0.91      0.87      2382
   Ballfault       0.97      0.92      0.94      2357
     Healthy       0.97      0.99      0.98      2392
  Innerfault       0.93      0.90      0.91      4869

    accuracy                           0.92     12000
   macro avg       0.93      0.93      0.93     12000
weighted avg       0.93      0.92      0.92     12000

F1 Score: 0.9239779092952728
```

CONFUSION MATRIX ANALYSIS

Displays counts of True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN) for each class.

- Classes: BallFault, Ballfault, Healthy, Innerfault.

Observations:

- Class BallFault: High recall (93%), indicating effective identification but potential false positives.
- Class Ballfault: Precision and recall consistently high (96% precision, 93% recall).
- Class Healthy: Almost perfect identification with recall at 99%.
- Class Innerfault: Strong precision (94%) but slightly lower recall (89%).

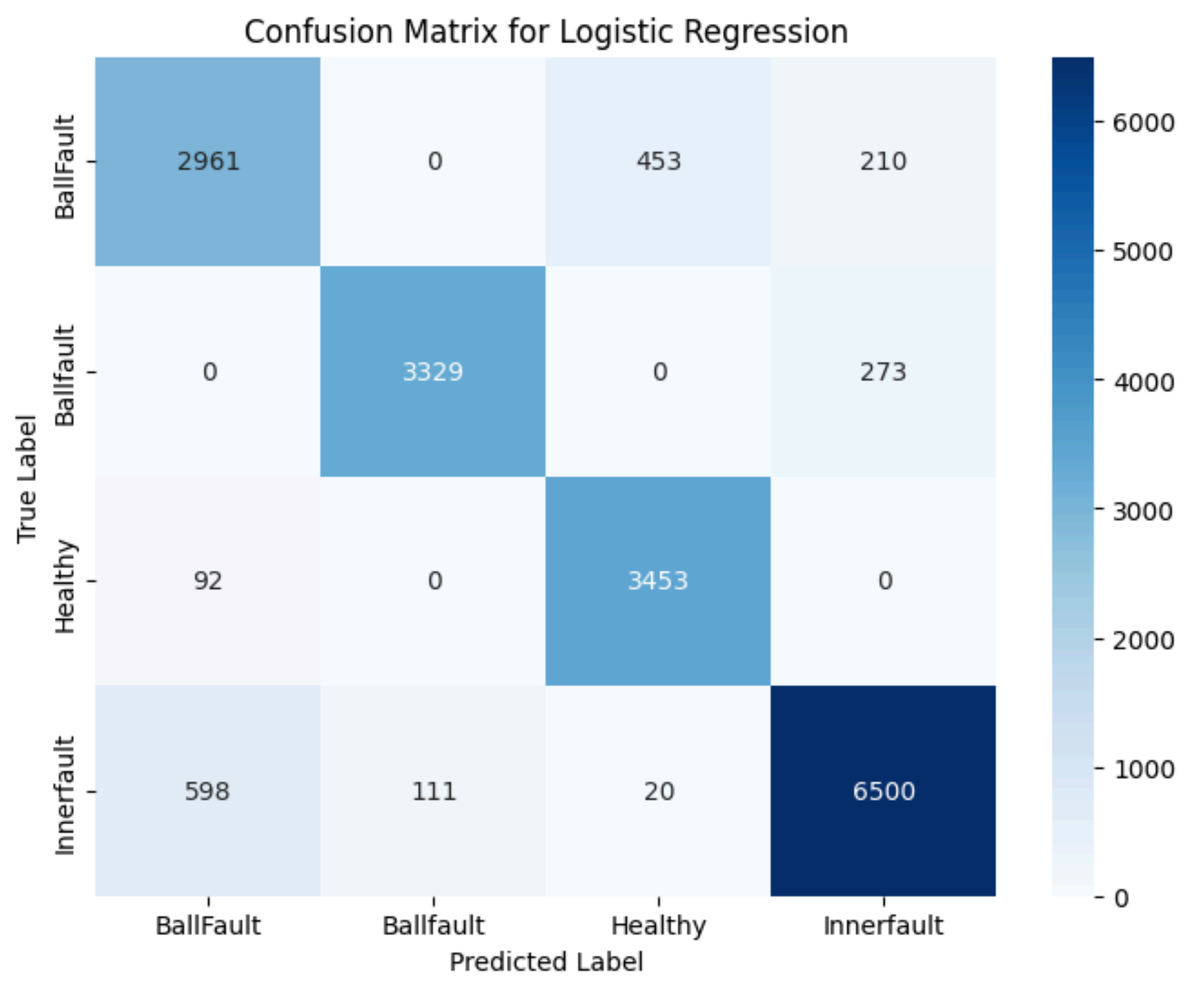
Metrics Derived:

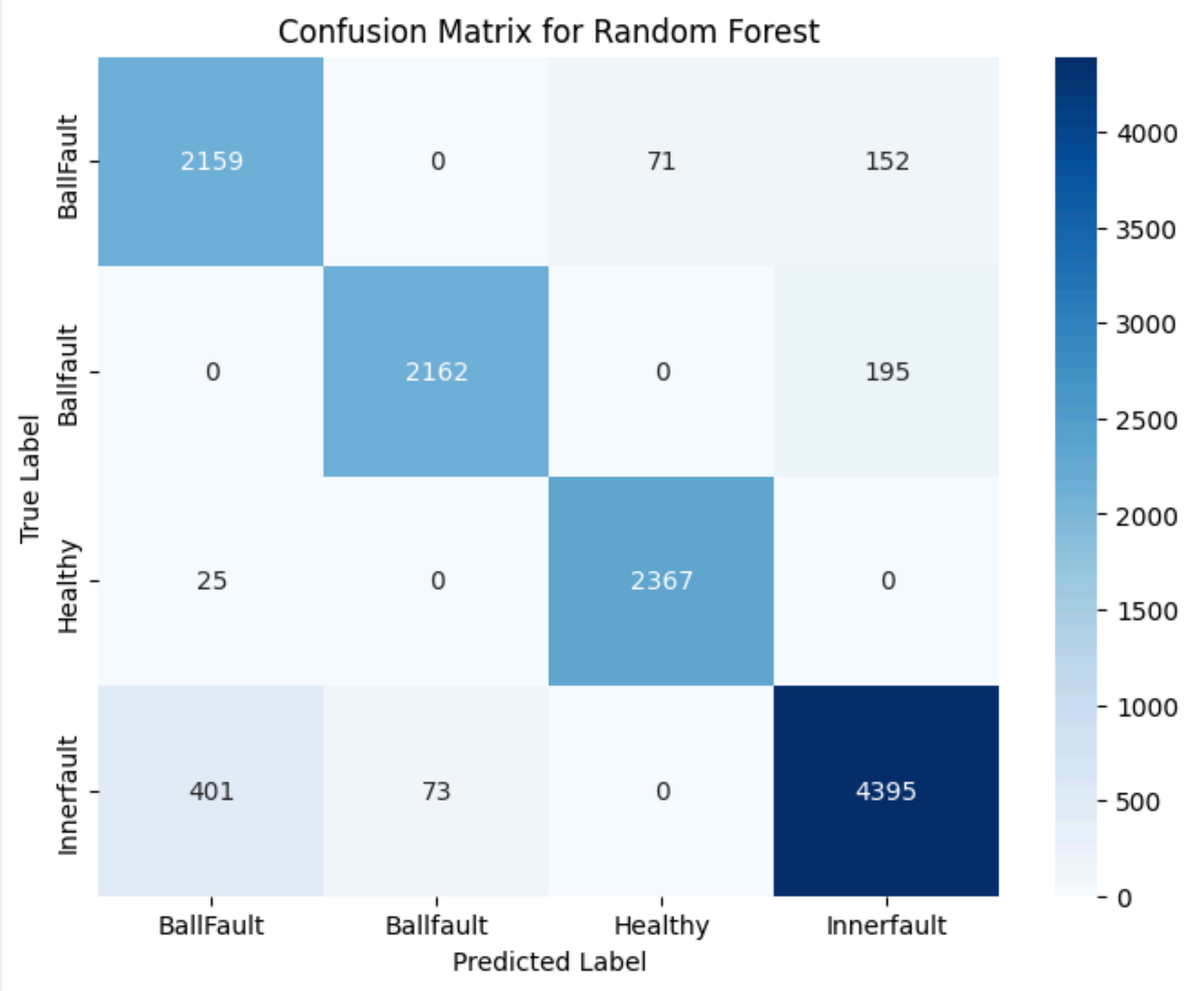
- Weighted Average F1 Score: 0.92 – strong overall performance.
- Misclassifications: Errors primarily observed between BallFault and Innerfault.

Visualization: Heatmap representation of confusion matrix.

- Diagonal dominance: Correct predictions visible in high diagonal values.
- Misclassification trends in off-diagonal cells for deeper insight.

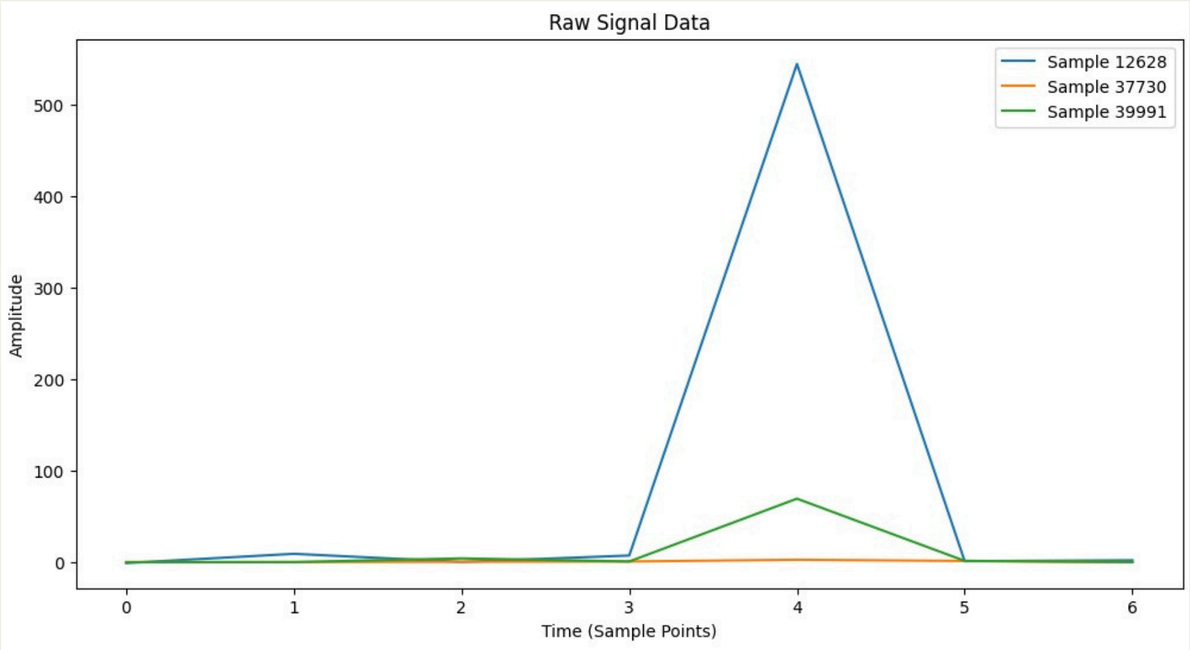
CONFUSION MATRIX ANALYSIS

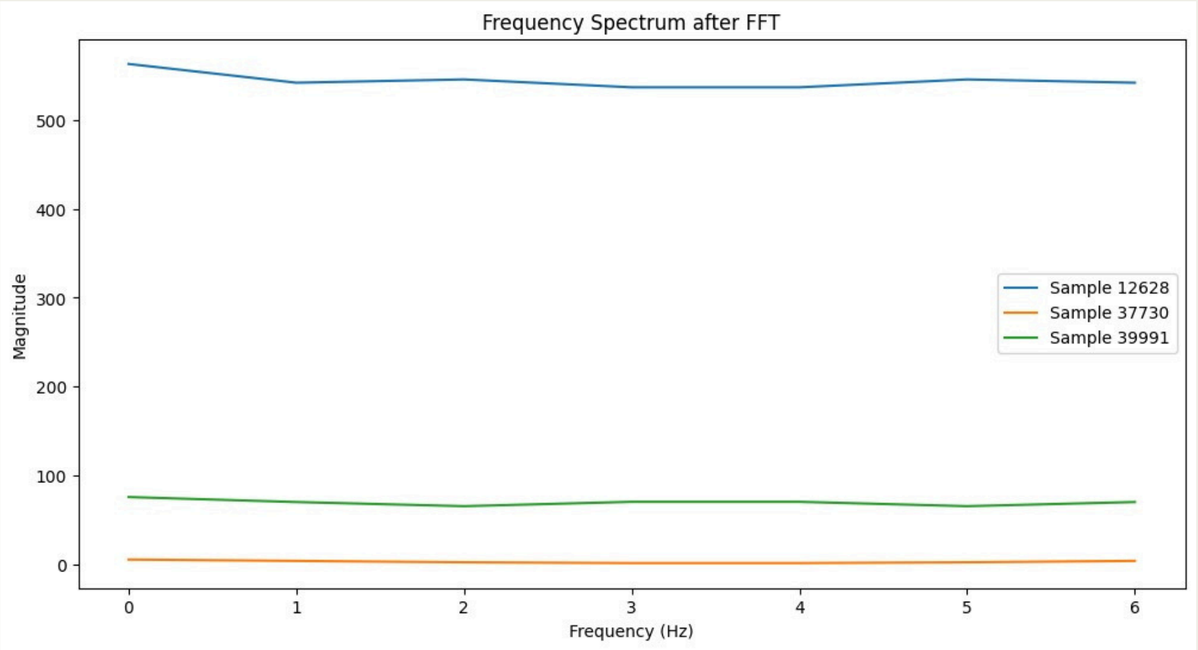




The number of faults misclassified as healthy reduced from 473 to 71 which was our primary goal.

PLOTS FOR ANALYSIS





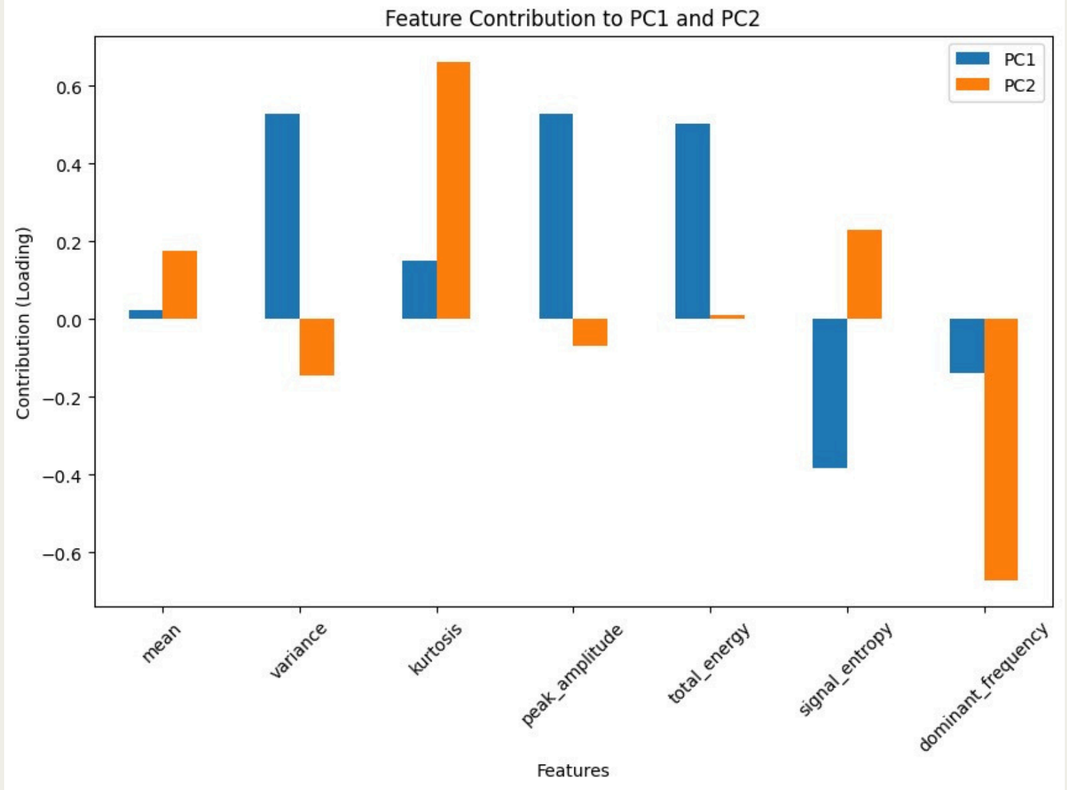
Raw Signal Data:

- Features: Amplitude over time, varying sample patterns.
- Behavior: Distinct spikes in certain samples indicate possible fault signals or transient events.

Frequency Spectrum after FFT:

- Features: Magnitude versus frequency, dominant frequency components.
- Behavior: Lower magnitudes suggest steady components, while variations point to faults or irregularities.

PLOTS FOR ANALYSIS



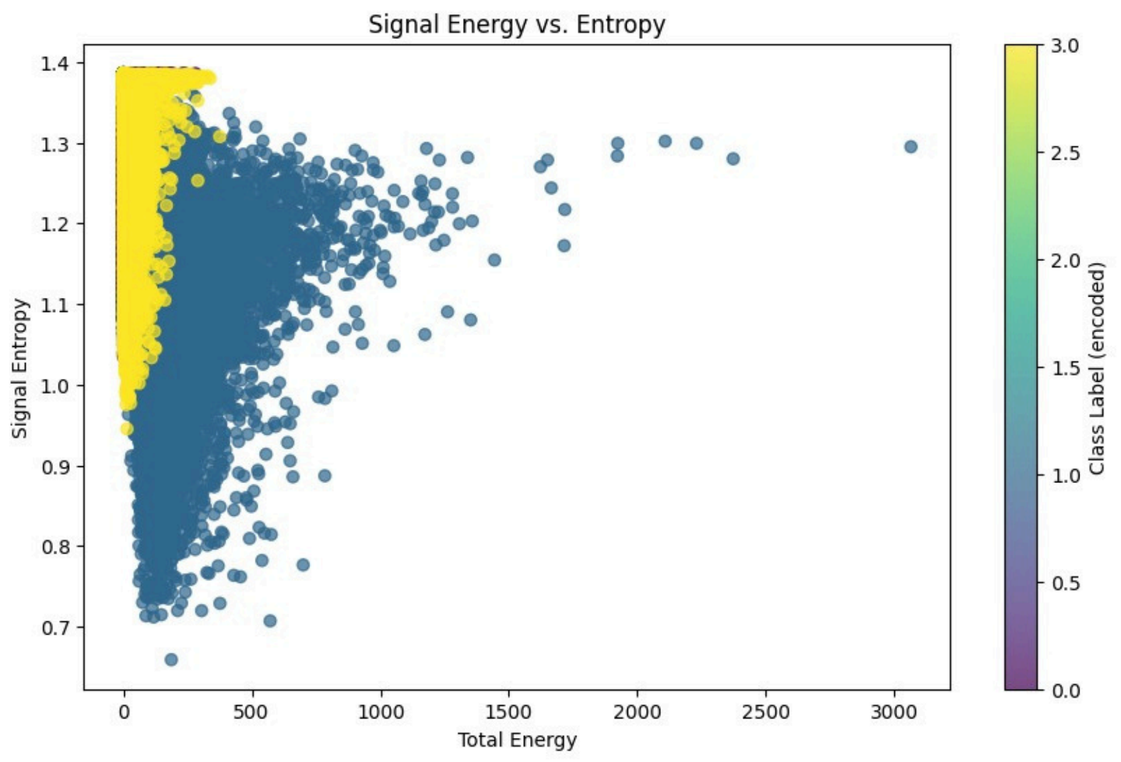
2D Visualization After Feature Selection:

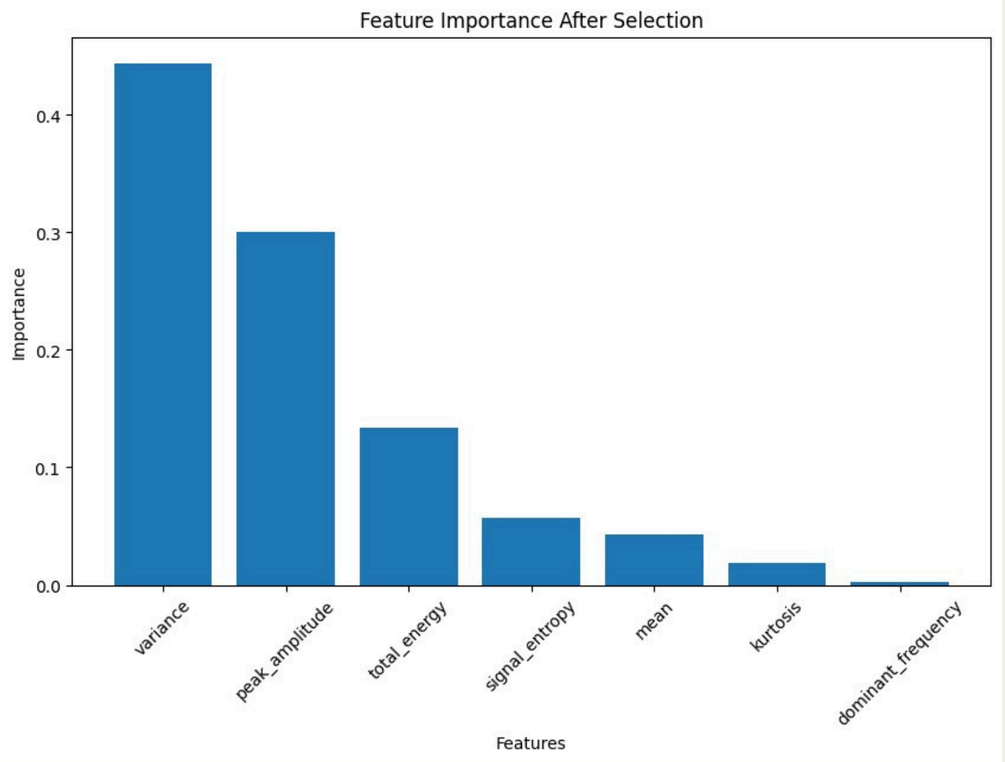
- PCA was applied to visualize the separability of selected features in a reduced 2D space, aiding interpretability.

Feature Contribution to PC1 and PC2:

- Bar chart represents how features contribute to PC1 and PC2 in PCA. Higher bars indicate greater influence on components

PLOTS FOR ANALYSIS





Signal Energy vs. Entropy:

- Features: Total signal energy, entropy, and encoded class labels.
- Behavior: Clusters show class separability; higher energy often corresponds to lower entropy in faulty signals.

Feature Importance After Selection:

- Highlights relative importance of features post-selection. Bars indicate the contribution of each feature to the model's performance.

Thank you!