Untitled0

May 18, 2025

1 Biomedical Signal Analysis Project

1.0.1 ECG Heart Rate Variability Study

1.0.2 Team Members:

Name	Student ID
Eslam Sameh Saeed Elsayed	20010298
Hassan Ibrahim Hassaan	20010484
Mohamed Ahmed Saad Ahmed	20011445

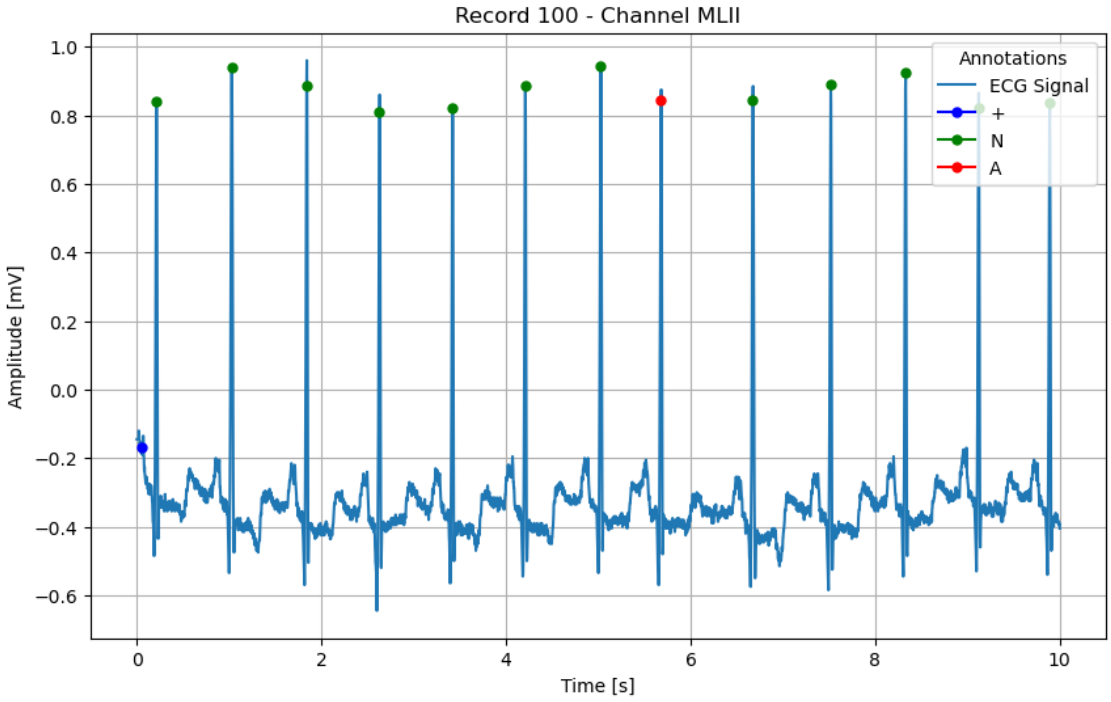
2 1. Dataset Exploration and Visualization

[3]:
```
import wfdb
import matplotlib.pyplot as plt
import numpy as np

# Load the ECG records from the current directory
record_100 = wfdb.rdrecord('100')
record_101 = wfdb.rdrecord('101')
record_102 = wfdb.rdrecord('102')

# Store the records and their corresponding names for easy looping
records = [(record_100, "100"), (record_101, "101"), (record_102, "102")]

# Define a color mapping for each annotation type
color_map = {
    '+': 'blue',
    'A': 'red',
    'N': 'green',
    '/': 'purple',
    'V': 'orange',
    'F': 'pink',
    'R': 'brown'
}

# Loop through each record and plot the ECG signal
```

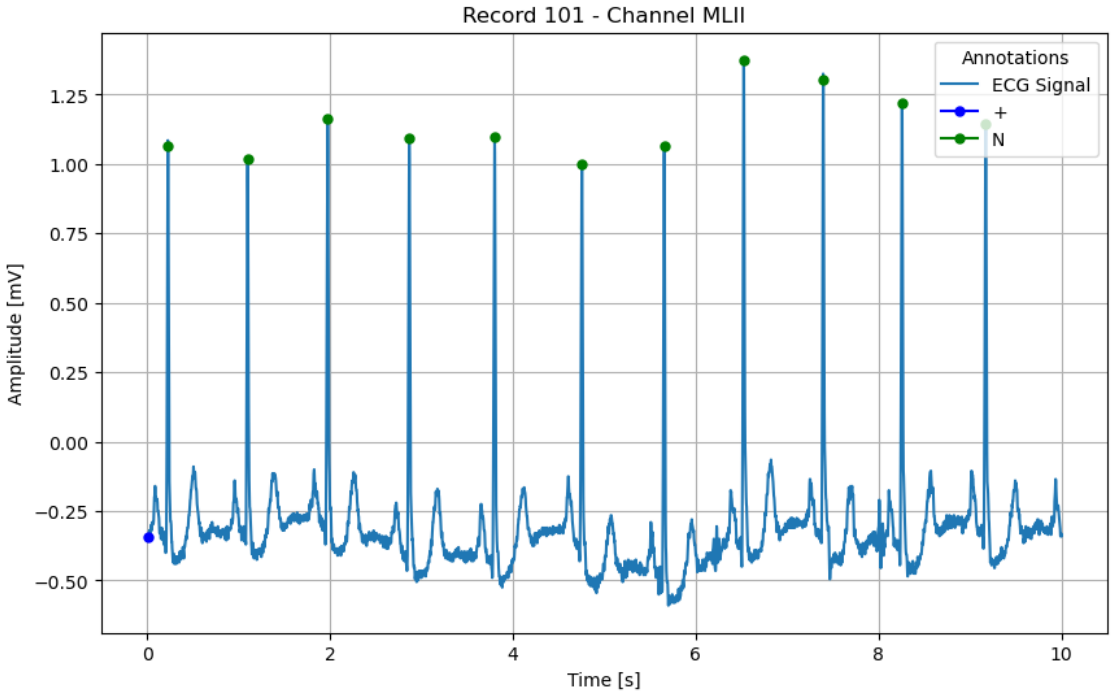
```
for i, (record, rec_name) in enumerate(records):
    # Load the annotation file for each record (contains heartbeat types and␣
 ↪positions)
    annotation = wfdb.rdann(f'{rec_name}', 'atr')

    fs = record.fs  # Sampling frequency
    samples = int(fs * 10)  # Number of samples for the first 10 seconds

    # Select the "MLII" channel if available, otherwise use the first available␣
 ↪channel
    if "MLII" in record.sig_name:
        ch_index = record.sig_name.index("MLII")
        ch_name = "MLII"
    else:
        ch_index = 0
        ch_name = record.sig_name[0]
        print(f" Channel 'MLII' not found in record {rec_name}, using␣
 ↪'{ch_name}' instead.")

    # Extract the signal data for the selected channel and time range
    signal = record.p_signal[:samples, ch_index]
    time_axis = np.linspace(0, 10, samples)  # Time axis for plotting (0 to 10␣
 ↪seconds)

    # Create a new figure for the current record
    plt.figure(figsize=(10, 6))

    # Plot the ECG signal
    plt.plot(time_axis, signal, label='ECG Signal')

    # Extract annotations (heartbeats) that occur within the first 10 seconds
    ann_indices = np.where(annotation.sample < samples)[0]
    ann_samples = annotation.sample[ann_indices]
    ann_symbols = [annotation.symbol[j] for j in ann_indices]

    # Plot each annotation as a dot on the signal with its symbol as the label␣
 ↪(once per symbol)
    plotted_labels = set()
    for j, sym in enumerate(ann_symbols):
        sample = ann_samples[j]
        time = sample / fs

        if sym not in plotted_labels:
            # Use the color from the color_map for each annotation type
            color = color_map.get(sym, 'black')  # Default to 'black' if the␣
 ↪annotation is not in the map
```

```
            plt.plot(time, signal[sample], marker='o', markersize=5, label=sym,␣
↪color=color)
            plotted_labels.add(sym)
        else:
            # Plot the annotation without adding to legend again
            color = color_map.get(sym, 'black')
            plt.plot(time, signal[sample], marker='o', markersize=5,␣
↪color=color)

    # Count and print the number of each heartbeat type in the first 10 seconds
    unique_heartbeat_types = np.unique(ann_symbols)
    heartbeat_count = {ht: ann_symbols.count(ht) for ht in␣
↪unique_heartbeat_types}
    print(f"\nHeartbeat Types Distribution for Record {rec_name} (First 10␣
↪seconds):")
    for ht, count in heartbeat_count.items():
        print(f"{ht}: {count} occurrences")

    # Add title and labels to the plot
    plt.title(f"Record {rec_name} - Channel {ch_name}")
    plt.xlabel("Time [s]")
    plt.ylabel("Amplitude [mV]")
    plt.grid(True)

    # Display a legend with unique annotation symbols
    plt.legend(loc='upper right', title="Annotations")

    # Show the plot
    plt.show()
```

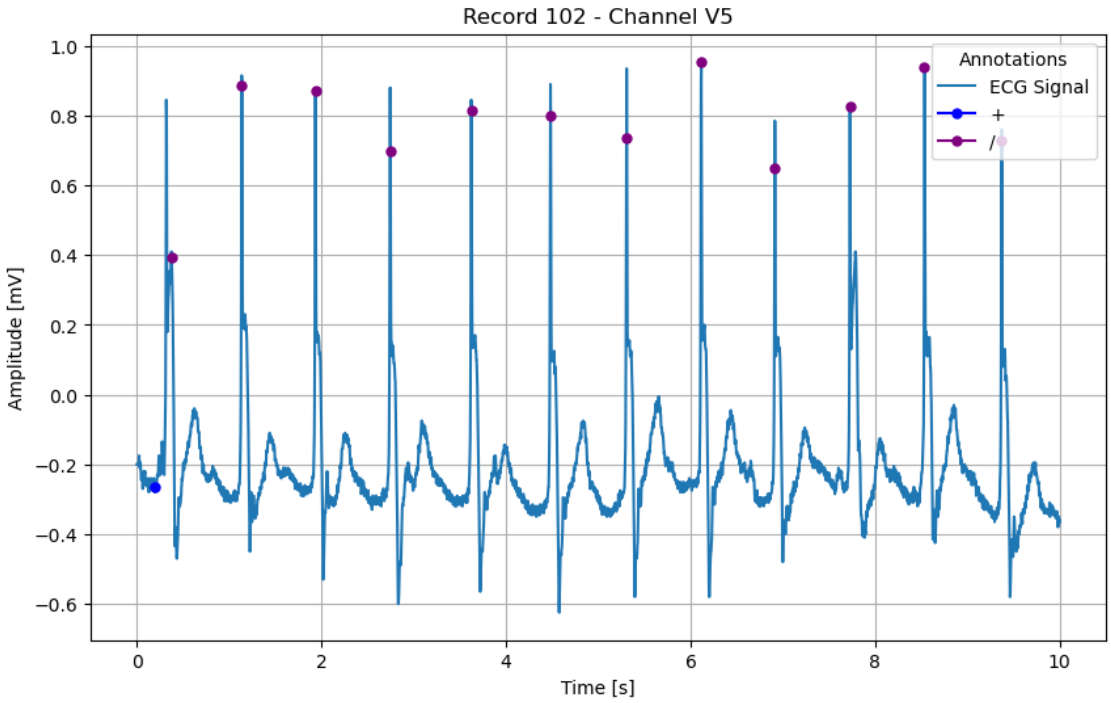
```
Heartbeat Types Distribution for Record 100 (First 10 seconds):
+: 1 occurrences
A: 1 occurrences
N: 12 occurrences
```




```
Heartbeat Types Distribution for Record 101 (First 10 seconds):
+: 1 occurrences
N: 11 occurrences
```



```
  Channel 'MLII' not found in record 102, using 'V5' instead.

Heartbeat Types Distribution for Record 102 (First 10 seconds):
+: 1 occurrences
/: 12 occurrences
```



2.1 Summary of Heartbeat Types (First 10 Seconds)

1.Record 100 : Normal (N) beats dominate (12 occurrences), indicating a mostly regular rhythm.

Atrial premature (A) and noise (+) appear once each, suggesting minor anomalies or artifacts.

2.Record 101: Normal (N) beats are predominant (11 occurrences).

A single noise artifact (+) is present, but no significant arrhythmias are detected in this segment.

3.Record 102: Paced beats (/) are the most frequent (12 occurrences), suggesting this record may come from a patient with an artificial pacemaker.

A single noise artifact (+) is present, but no natural beats (N) appear in this segment.

2.2 Key Observations

- Record 100 and 101 show primarily normal sinus rhythm with rare interruptions.
- Record 102 is distinctly different, likely due to paced rhythms (common in pacemaker patients).

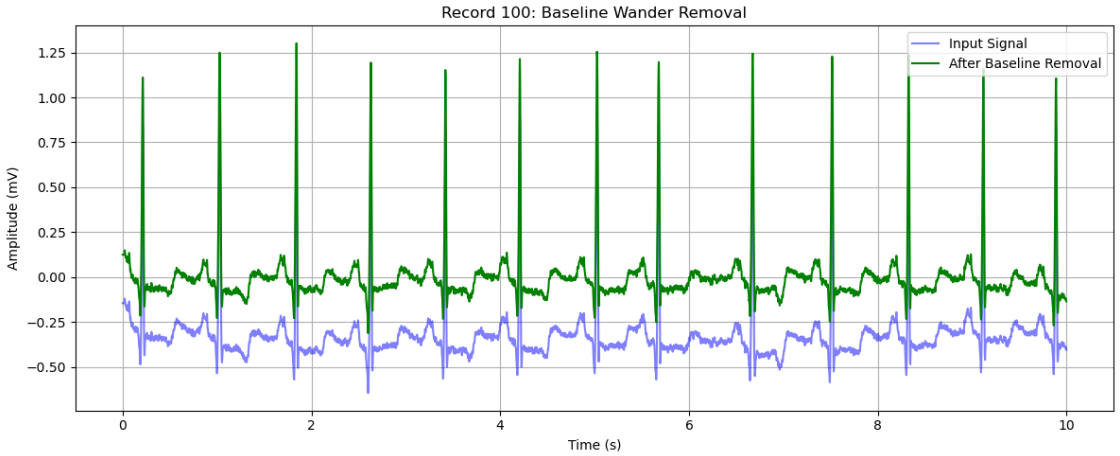
3 2. Signal Preprocessing and Noise Removal

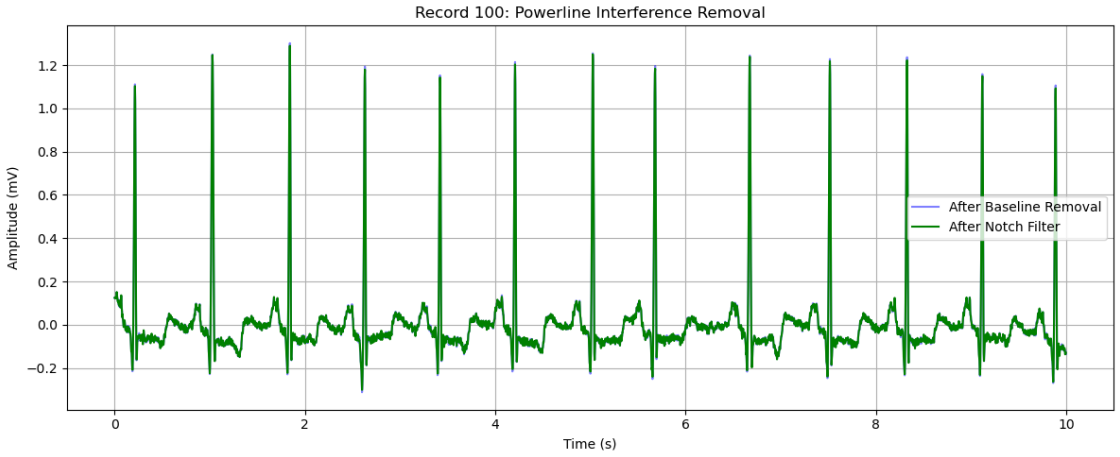
- List item
- List item

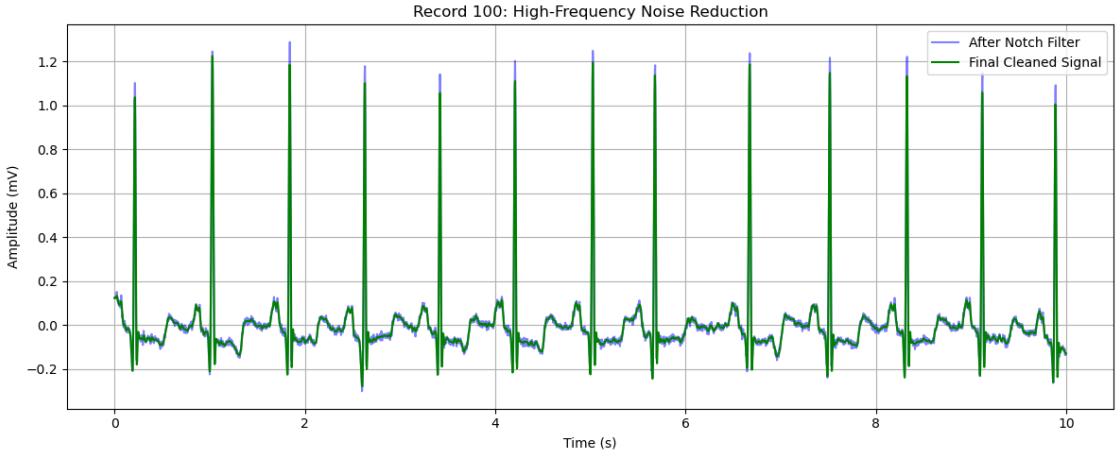
[4]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal as sp
import wfdb

# Records to process
records = ['100', '101', '102']

# Define preprocessing functions
def remove_baseline_wander(signal, fs, cutoff=0.5):
    n = len(signal)
    fft_signal = np.fft.fft(signal)
    freqs = np.fft.fftfreq(n, d=1/fs)
    fft_signal[np.abs(freqs) < cutoff] = 0
    return np.fft.ifft(fft_signal).real

def notch_filter(signal, fs, freq=50, Q=30):
    nyquist = 0.5 * fs
    b, a = sp.iirnotch(freq / nyquist, Q)
    return sp.filtfilt(b, a, signal)

def lowpass_filter(signal, fs, cutoff=40, order=4):
    nyquist = 0.5 * fs
    b, a = sp.butter(order, cutoff / nyquist, btype='low')
    return sp.filtfilt(b, a, signal)

# Process each record
for rec_name in records:
    # Load record
    record = wfdb.rdrecord(rec_name)
    fs = record.fs

    # Select channel (MLII if available, otherwise first channel)
    if 'MLII' in record.sig_name:
        ch_idx = record.sig_name.index('MLII')
    else:
        ch_idx = 0
```

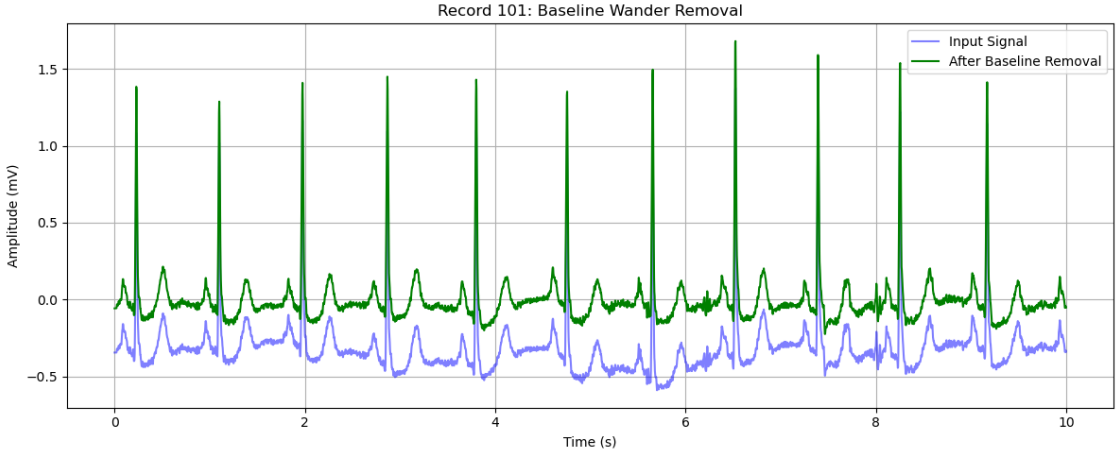
```
    ecg_signal = record.p_signal[:, ch_idx]

    # Extract 10-second segment
    samples = int(fs * 10)
    ecg_signal = ecg_signal[:samples]
    time = np.arange(samples) / fs

    # Preprocessing pipeline
    ecg_baseline_removed = remove_baseline_wander(ecg_signal, fs)
    ecg_notch_filtered = notch_filter(ecg_baseline_removed, fs)
    ecg_clean = lowpass_filter(ecg_notch_filtered, fs)

    # Create comparison figures with consistent colors
    plt.figure(figsize=(12, 5))
    plt.plot(time, ecg_signal, 'b', alpha=0.5, label='Input Signal')
    plt.plot(time, ecg_baseline_removed, 'g', linewidth=1.5, label='After␣
↪Baseline Removal')
    plt.title(f'Record {rec_name}: Baseline Wander Removal')
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude (mV)')
    plt.legend()
    plt.grid()
    plt.tight_layout()
    plt.show()

    plt.figure(figsize=(12, 5))
    plt.plot(time, ecg_baseline_removed, 'b', alpha=0.5, label='After Baseline␣
↪Removal')
    plt.plot(time, ecg_notch_filtered, 'g', linewidth=1.5, label='After Notch␣
↪Filter')
    plt.title(f'Record {rec_name}: Powerline Interference Removal')
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude (mV)')
    plt.legend()
    plt.grid()
    plt.tight_layout()
    plt.show()

    plt.figure(figsize=(12, 5))
    plt.plot(time, ecg_notch_filtered, 'b', alpha=0.5, label='After Notch␣
↪Filter')
    plt.plot(time, ecg_clean, 'g', linewidth=1.5, label='Final Cleaned Signal')
    plt.title(f'Record {rec_name}: High-Frequency Noise Reduction')
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude (mV)')
    plt.legend()
    plt.grid()
```

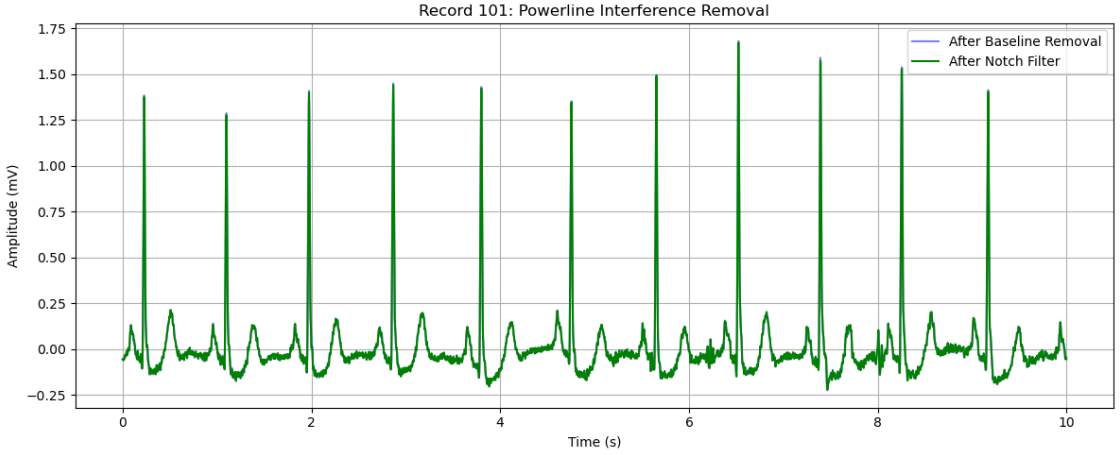
```
    plt.tight_layout()
    plt.show()
```

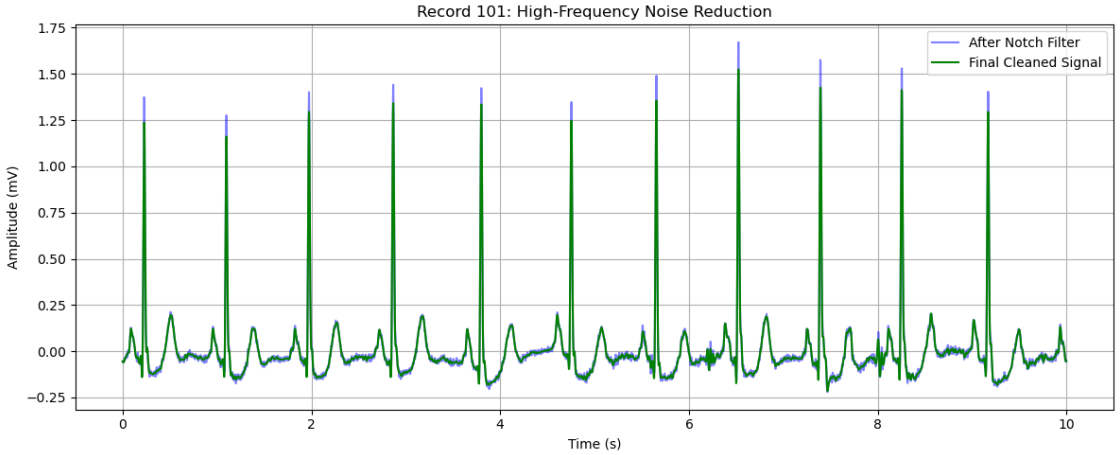


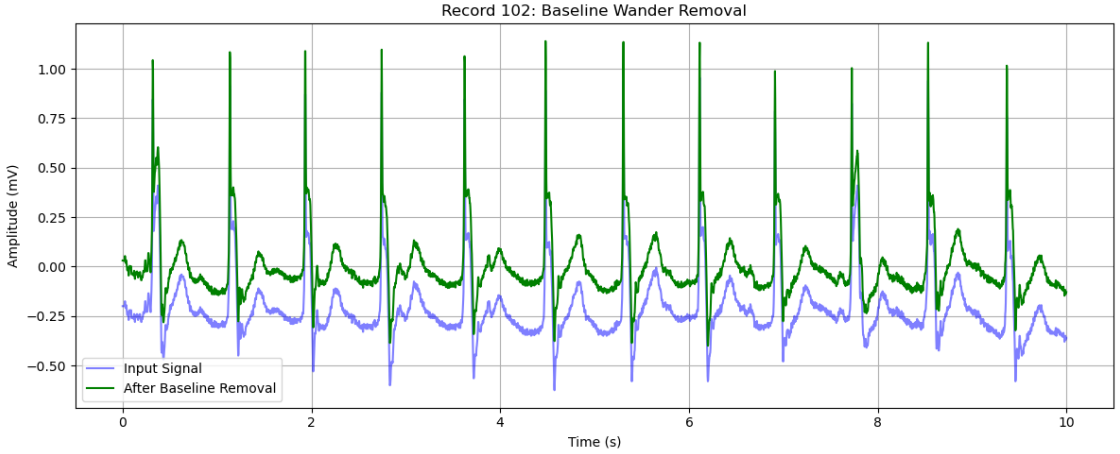


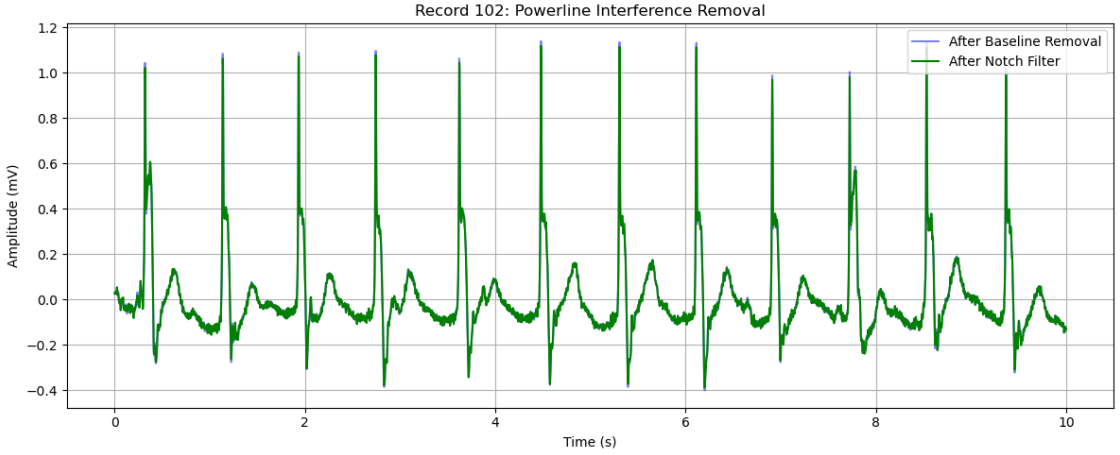


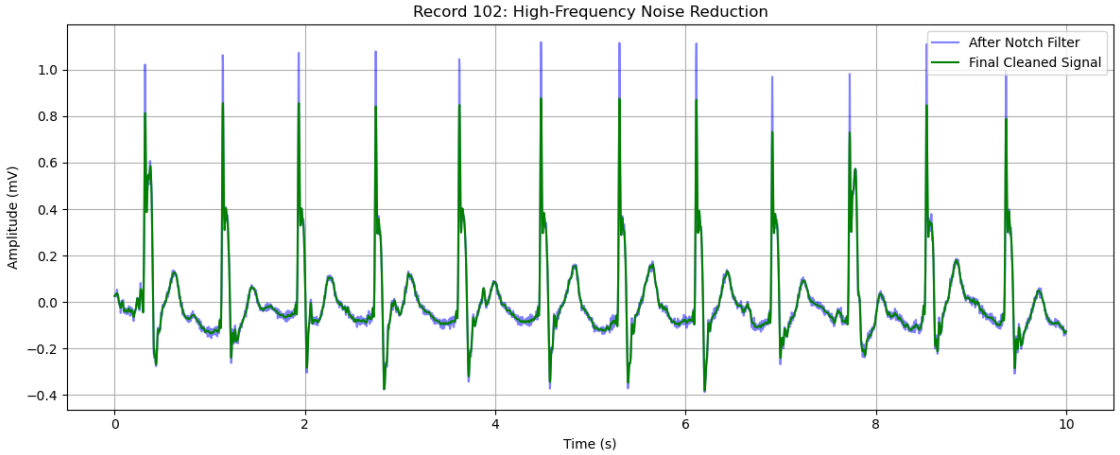

Record 101: Baseline Wander Removal
Input Signal
After Baseline Removal
Amplitude (mV)
Time (s)


Record 101: Powerline Interference Removal
After Baseline Removal
After Notch Filter
Amplitude (mV)
Time (s)


Record 101: High-Frequency Noise Reduction
After Notch Filter
Final Cleaned Signal
Amplitude (mV)
Time (s)


Record 102: Baseline Wander Removal
Amplitude (mV)
Time (s)
Input Signal
After Baseline Removal


Record 102: Powerline Interference Removal
Amplitude (mV)
Time (s)
After Baseline Removal
After Notch Filter


Record 102: High-Frequency Noise Reduction
Amplitude (mV)
Time (s)
After Notch Filter
Final Cleaned Signal

4 Observation

4.0.1 1. Baseline Wander Removal (0.5Hz High-pass Filter)

- **Signal Centering**: The signal becomes properly centered around zero (removes slow drifts)
- **P-QRS-T Preservation**: All waveform features remain intact but "float" less

4.0.2 2. Notch Filter (50Hz Removal)

- **Subtle Effect:** In clean MIT-BIH records, changes are minimal (visible only in zoomed view)

4.0.3 3.Low-pass Filter (40Hz Cutoff)

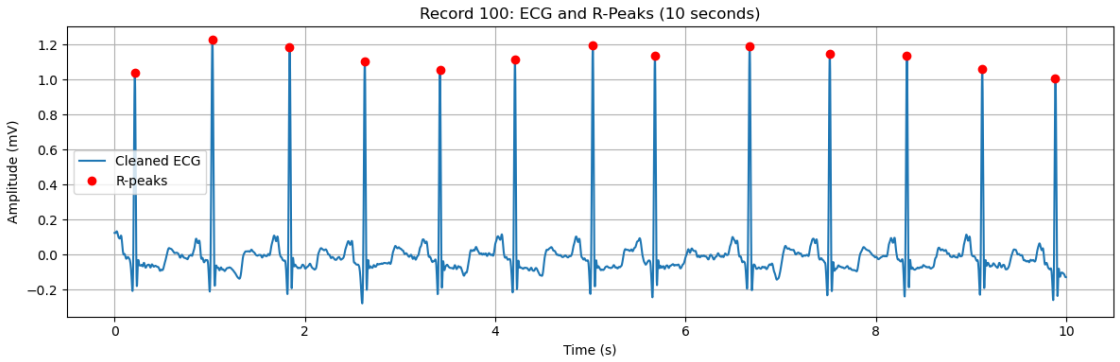
- **Noise Reduction**: High-frequency muscle noise/artifacts are smoothed
- **P/T Wave Effects**: Slight rounding of high-frequency components in P/T waves

5 3. R-Peak Detection and Heart Rate Calculation

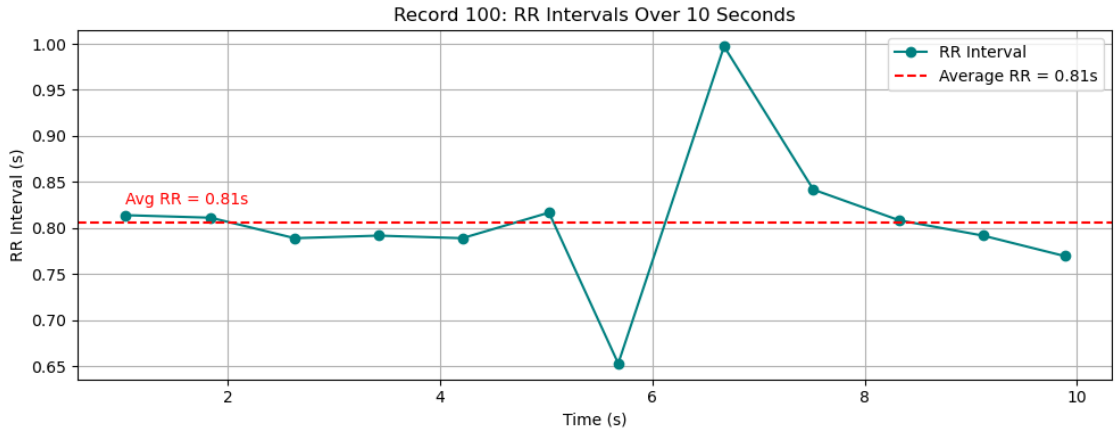
[16]:
```
from scipy.signal import find_peaks
import matplotlib.pyplot as plt
import numpy as np
import wfdb

# Reuse preprocessing functions here:
# remove_baseline_wander, notch_filter, lowpass_filter

records = ['100', '101', '102']

for rec_name in records:
    record = wfdb.rdrecord(rec_name)
    fs = record.fs

    ch_idx = record.sig_name.index('MLII') if 'MLII' in record.sig_name else 0
    ecg_signal = record.p_signal[:, ch_idx]

    samples = int(fs * 10)
    ecg_signal = ecg_signal[:samples]
    time = np.arange(samples) / fs

    ecg_clean = lowpass_filter(notch_filter(remove_baseline_wander(ecg_signal,␣
 ↪fs), fs), fs)

    peak_height = 0.4 * np.max(np.abs(ecg_clean))
    min_distance = int(0.25 * fs)
    peaks, _ = find_peaks(ecg_clean, height=peak_height, distance=min_distance)
```

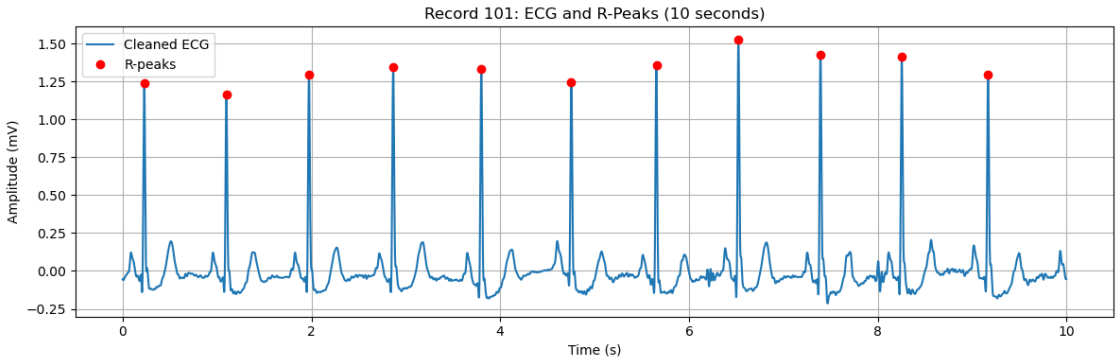
```
    # ECG with R-peaks
    plt.figure(figsize=(12, 4))
    plt.plot(time, ecg_clean, label="Cleaned ECG")
    plt.plot(time[peaks], ecg_clean[peaks], 'ro', label="R-peaks")
    plt.title(f'Record {rec_name}: ECG and R-Peaks (10 seconds)')
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude (mV)')
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
    plt.show()

    rr_intervals = np.diff(peaks) / fs
    valid_indices = (rr_intervals > 0.3) & (rr_intervals < 2.0)
    rr_intervals = rr_intervals[valid_indices]
    rr_times = (time[peaks][1:])[valid_indices]

    # Plot RR intervals over time with average line
    plt.figure(figsize=(10, 4))
    plt.plot(rr_times, rr_intervals, marker='o', linestyle='-', color='teal',␣
↪label='RR Interval')

    if len(rr_intervals) > 0:
        mean_rr = np.mean(rr_intervals)
        heart_rate = 60 / mean_rr

        # Add average line and text
        plt.axhline(mean_rr, color='red', linestyle='--', label=f'Average RR =␣
↪{mean_rr:.2f}s')
        plt.text(rr_times[0], mean_rr + 0.02, f'Avg RR = {mean_rr:.2f}s',␣
↪color='red')

        print(f"\n Record {rec_name}: Average Heart Rate = {heart_rate:.2f}␣
↪BPM")

        if heart_rate < 60:
            print(" Observation: Possible Bradycardia (Low heart rate)")
        elif heart_rate > 100:
            print(" Observation: Possible Tachycardia (High heart rate)")
        else:
            print(" Observation: Normal heart rate")
    else:
        print(f"\n Record {rec_name}: No valid RR intervals detected.")

    plt.title(f'Record {rec_name}: RR Intervals Over 10 Seconds')
    plt.xlabel('Time (s)')
```

```
    plt.ylabel('RR Interval (s)')
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
    plt.show()
```

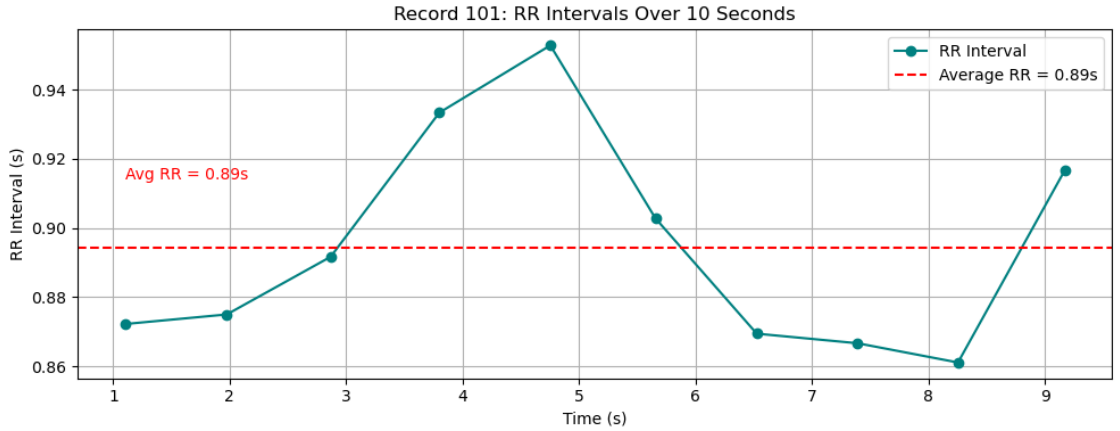


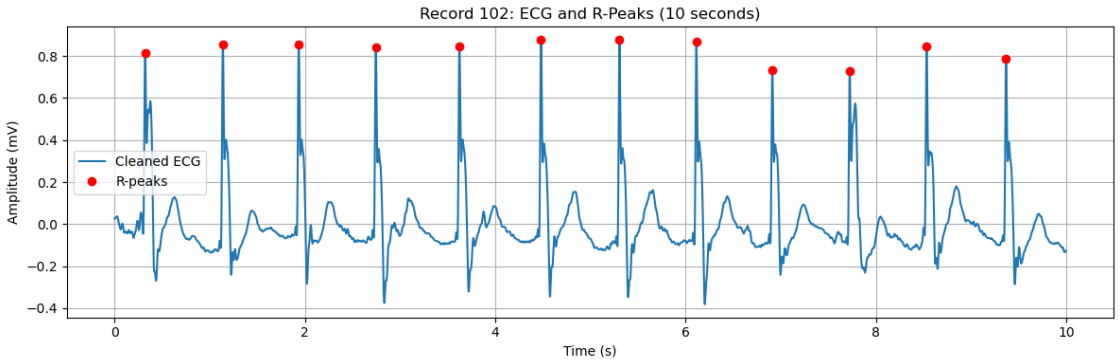

```
Record 100: Average Heart Rate = 74.44 BPM
Observation: Normal heart rate
```





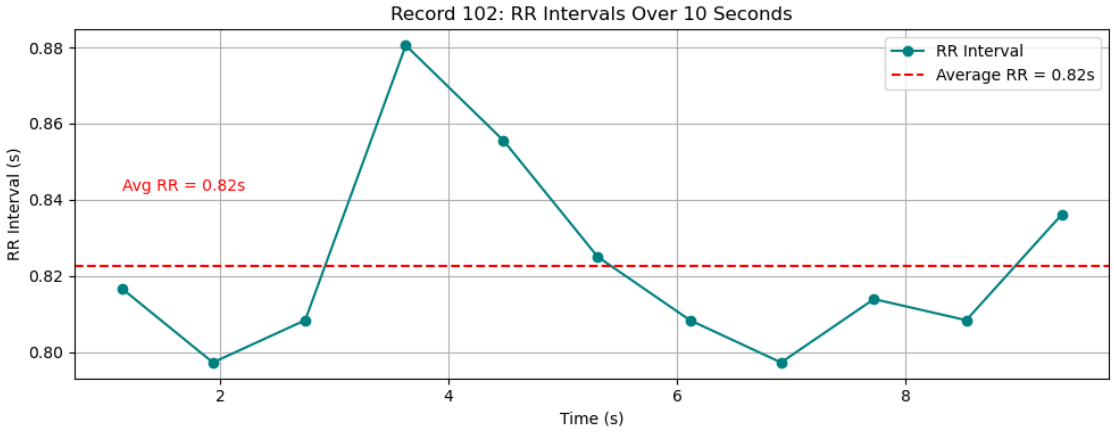
Record 101: Average Heart Rate = 67.10 BPM
Observation: Normal heart rate





Record 102: Average Heart Rate = 72.95 BPM

```
Observation: Normal heart rate
```



6 Detailed Comments on Each ECG Record

6.1 1. Record 100

Avg RR Interval: 0.81s → Heart Rate: ~74 bpm (normal range).

Key Observations:

- Stable rhythm with minor fluctuations in RR intervals (0.75–0.95s).
- No significant irregularity; typical of a healthy, resting heart.
- Slight variability may reflect normal autonomic adjustments (e.g., breathing, mild activity).

6.2 2. Record 101

Avg RR Interval: 0.89s → Heart Rate: ~67 bpm (normal/slightly slow).

Key Observations:

- Gradual shortening of RR intervals (0.94s → 0.86s) suggests respiratory sinus arrhythmia—a benign, physiological phenomenon where HR increases with inhalation and decreases with exhalation.
- Common in young, healthy individuals and athletes.
- No clinical concern unless accompanied by symptoms (e.g., dizziness).

7 3. Record 102

Avg RR Interval: 0.82s → Heart Rate: ~73 bpm (normal range).

Key Observations:

- Highly consistent RR intervals (avg 0.82s) indicate a regular, steady rhythm.
- Reflects a relaxed state with minimal autonomic fluctuation.
- Ideal example of a "textbook" normal sinus rhythm.

---

8 4. Arrhythmia Detection and Classification

[1]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, ConfusionMatrixDisplay,␣
 ↪RocCurveDisplay
from sklearn.metrics import (
    confusion_matrix,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    roc_curve,
    auc
)
import matplotlib.pyplot as plt
import seaborn as sns
```

[13]:
```
def extract_hrv_features(rr_intervals):
    """Extract Heart Rate Variability features from RR intervals"""
    features = {}

    # Time-domain features
    features['mean_rr'] = np.mean(rr_intervals)
    features['std_rr'] = np.std(rr_intervals)
    features['rmssd'] = np.sqrt(np.mean(np.square(np.diff(rr_intervals))))
    features['nn50'] = np.sum(np.abs(np.diff(rr_intervals)) > 0.05)
    features['pnn50'] = features['nn50'] / len(rr_intervals) * 100

    # Frequency-domain features would require Lomb-Scargle periodogram
    # (omitted for simplicity but important for comprehensive HRV analysis)

    return features

remove_powerline_noise = notch_filter
apply_bandpass_filter  = lowpass_filter
```

```
def load_labeled_data(records, window_size=30):
    X, y = [], []
    for record in records:
        signals, fields = wfdb.rdsamp(str(record))
        ann = wfdb.rdann(str(record), 'atr')
        try:
            mlii_idx = fields['sig_name'].index('MLII')
        except ValueError:
            mlii_idx = 0
        fs = fields['fs']

        for start in range(0, len(signals) - window_size*fs, window_size*fs//2):
            end = start + window_size*fs
            seg = signals[start:end, mlii_idx]

            # preprocessing
            proc = remove_baseline_wander(seg, fs)
            proc = remove_powerline_noise(proc, fs)    #  notch_filter
            proc = apply_bandpass_filter(proc, fs)     #  lowpass_filter

            try:
                r_peaks = detect_r_peaks(proc, fs)
                if len(r_peaks) < 10: continue

                rr = np.diff(r_peaks) / fs
                feats = extract_hrv_features(rr)

                X.append(list(feats.values()))
                y.append(label)
            except Exception as e:
                print(f"Error on {record} window {start}-{end}: {e}")
    return np.array(X), np.array(y)

# Records to use for training/testing (using a subset for demonstration)
train_records = [100, 101, 103, 105, 106, 107, 108, 109, 111, 112, 116, 117,␣
 ↪118, 119, 121, 122, 123, 124, 200, 201, 202, 203, 205, 207, 208, 209, 210,␣
 ↪212, 213, 214, 215, 217, 219, 220, 221, 222, 223, 228, 230, 231, 232, 233,␣
 ↪102, 104, 113, 114, 115]

test_records = [102, 104, 113, 114, 115]
```

[ ]:
```
x_train, y_train = load_labeled_data(train_records)
print("x_train.shape:", x_train.shape)
print("y_train.shape:", y_train.shape)
print(f"Training data:{x_train.shape[0]} samples")
x_test, y_test = load_labeled_data(test_records)
```

```
print(f"Training data:{x_test.shape[0]} samples")

scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.fit_transform(x_test)

# Train Random Forest classifier
print("\nTraining classifier...")
clf = RandomForestClassifier(n_estimators=1000, criterion="entropy",␣
 ↪random_state=41)
clf.fit(x_train_scaled, y_train)

# Evaluate on test set
y_pred = clf.predict(x_test_scaled)
y_proba = clf.predict_proba(x_test_scaled)[:, 1]  # Probabilities for ROC

# Performance metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
print("Classification Report:")
print(classification_report(y_test, y_pred, target_names=["Normal","Abnormal"]))
# Confusion Matrix
ConfusionMatrixDisplay.from_predictions(y_test, y_pred,
    display_labels=["Normal","Abnormal"], cmap="Blues")
plt.title("Confusion Matrix")
plt.show()

# ROC Curve
RocCurveDisplay.from_predictions(y_test, y_proba)
plt.title(f"ROC Curve (AUC = {auc(*roc_curve(y_test, y_proba)[:2]):.2f})")
plt.show()

# Feature Importances
importances = clf.feature_importances_
feat_names = ['mean_rr','std_rr','rmssd','nn50','pnn50']
sorted_idx = np.argsort(importances)[::-1]
plt.bar([feat_names[i] for i in sorted_idx], importances[sorted_idx])
plt.title("Feature Importances")
plt.ylabel("Importance")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

[ ]: