MIT-BIH noise stress test database

October 18, 2025

[31]:
```python
# Step 1: Import libraries
import os
import wfdb
import pandoc
import zipfile
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.inspection import permutation_importance
from scipy.signal import find_peaks
from scipy.stats import skew, kurtosis
from sklearn.impute import SimpleImputer
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score, roc_auc_score, confusion_matrix, roc_curve, auc,␣
 ↪classification_report
```

[3]:
```python
base_path = "/Users/mohithreddy/mit-bih-noise-stress-test-database-1.0.0/
 ↪mit-bih-noise-stress-test-database-1.0.0"
```

[4]:
```python
zip_path = "/Users/mohithreddy/Downloads/mit-bih-noise-stress-test-database-1.0.
 ↪0.zip"  # change if needed
extract_dir = "/Users/mohithreddy/mit-bih-noise-stress-test-database-1.0.0"

# Extract the zip
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_dir)

print("Extracted to:", extract_dir)
print("Files inside:", os.listdir(extract_dir)[:10])
```

```
Extracted to: /Users/mohithreddy/mit-bih-noise-stress-test-database-1.0.0
Files inside: ['mit-bih-noise-stress-test-database-1.0.0', '__MACOSX']
```

[5]:
```python
os.getcwd()
```

[5]: '/Users/mohithreddy'

[6]:
```python
os.listdir("mit-bih-noise-stress-test-database-1.0.0")[:10]
```

[6]: ['mit-bih-noise-stress-test-database-1.0.0', '__MACOSX']

[7]:
```python
data_path = "mit-bih-noise-stress-test-database-1.0.0/
 ↪mit-bih-noise-stress-test-database-1.0.0"
```

[8]:
```python
os.listdir(data_path)[:10]
```

[8]:
```
['119e12.hea',
 '119e06.hea',
 'bw.hea',
 'em.hea-',
 'em.xws',
 'ma.dat',
 '118e12.xws',
 '118e06.xws',
 'RECORDS',
 '118e24.atr']
```

[9]:
```python
data_path = "/Users/mohithreddy/Downloads/mit-bih-noise-stress-test-database-1.
 ↪0.0/mit-bih-noise-stress-test-database-1.0.0"
```

[10]:
```python
# Show available header files
hea_files = [f for f in os.listdir(extract_dir) if f.endswith('.hea')]
print("Total .hea files:", len(hea_files))
print(hea_files[:15])  # show first 15
```

```
Total .hea files: 0
[]
```

[11]:
```python
data_path = "mit-bih-noise-stress-test-database-1.0.0/
 ↪mit-bih-noise-stress-test-database-1.0.0"

# Select 5 subjects
subjects = ['118e_6', '118e12', '118e18', '119e_6', '119e24']

records = {}

for subj in subjects:
    record_path = os.path.join(data_path, subj)
    record = wfdb.rdrecord(record_path)
    annotation = wfdb.rdann(record_path, 'atr')  # apnea annotations
```

```
    records[subj] = {
        "signal": record.p_signal[:, 0],
        "fs": record.fs,
        "annotations": annotation.sample,
        "labels": annotation.symbol
    }

    print(f"{subj}: Loaded with {len(record.p_signal)} samples at {record.fs}␣
↪Hz.")
```

```
118e_6: Loaded with 650000 samples at 360 Hz.
118e12: Loaded with 650000 samples at 360 Hz.
118e18: Loaded with 650000 samples at 360 Hz.
119e_6: Loaded with 650000 samples at 360 Hz.
119e24: Loaded with 650000 samples at 360 Hz.
```

[12]:
```
def label_class(ann):
    """
    Map annotation symbols to binary labels:
    Normal (0) and Abnormal (1)
    """
    # Normal beat types according to AAMI standards
    normal_symbols = ['N', 'L', 'R', 'e', 'j']

    labels = []
    for symbol in ann.symbol:
        if symbol in normal_symbols:
            labels.append(0)  # Normal
        else:
            labels.append(1)  # Abnormal
    return np.array(labels)
```

[13]:
```
def label_class_from_symbols(symbols):
    """Convert annotation symbols to binary labels: 0=Normal, 1=Abnormal"""
    normal_symbols = ['N', 'L', 'R', 'e', 'j']
    return np.array([0 if sym in normal_symbols else 1 for sym in symbols])

def create_window_targets(records, window_size_sec=3, overlap_sec=0.5):
    label_data = []

    for subj, rec in records.items():
        signal = rec["signal"]
        fs = rec["fs"]

        # --- handle both dictionary styles safely ---
        if "annotation" in rec:
            ann = rec["annotation"]
            beat_symbols = ann.symbol
```

```python
            beat_positions = np.array(ann.sample)
        elif "annotations" in rec and "labels" in rec:
            # old format from your earlier code
            beat_symbols = rec["labels"]
            beat_positions = np.array(rec["annotations"])
        else:
            raise KeyError(f" Subject {subj} has no annotation data!")

        # Convert beat types to 0/1 labels
        beat_labels = label_class_from_symbols(beat_symbols)

        # --- create windows ---
        win_len = int(fs * window_size_sec)
        step = int(fs * (window_size_sec - overlap_sec))

        for start in range(0, len(signal) - win_len, step):
            end = start + win_len
            beats_in_window = beat_labels[(beat_positions >= start) &␣
 ↪(beat_positions < end)]
            target = 1 if np.any(beats_in_window == 1) else 0
            label_data.append({
                "Subject_ID": subj,
                "Start": start,
                "End": end,
                "Target": target
            })

        print(f" Processed {subj}: {len(label_data)} windows labeled so far")

    return pd.DataFrame(label_data)


# --- Run the function ---
target_df = create_window_targets(records, window_size_sec=3, overlap_sec=0.5)

print("\n Target labels created successfully!")
print(target_df["Target"].value_counts())
```

```
 Processed 118e_6: 722 windows labeled so far
 Processed 118e12: 1444 windows labeled so far
 Processed 118e18: 2166 windows labeled so far
 Processed 119e_6: 2888 windows labeled so far
 Processed 119e24: 3610 windows labeled so far

 Target labels created successfully!
Target
0    2232
1    1378
```

```
Name: count, dtype: int64
```

[14]:
```python
# --- Define the labeling function if not already done ---
normal_class = ['N']
abnormal_class = ['A', 'V', 'L', 'R', 'E']

def label_beats(annotation):
    """Return binary labels for MIT-BIH arrhythmia beats."""
    labels = []
    for sym in annotation.symbol:
        if sym in normal_class:
            labels.append(0)    # Normal
        elif sym in abnormal_class:
            labels.append(1)    # Abnormal
        else:
            labels.append(np.nan)  # Ignore other types
    return np.array(labels)


# --- Check labels for all selected subjects ---
subjects = ['118e_6', '118e12', '118e18', '119e_6', '119e24']

for rec in subjects:
    ann = wfdb.rdann(os.path.join(data_path, rec), "atr")  # load annotation␣
 ↪file
    labels = label_class(ann)
    unique, counts = np.unique(labels[~np.isnan(labels)], return_counts=True) ␣
 ↪# ignore NaN
    print(f"Subject {rec} - Label Distribution:")
    for u, c in zip(unique, counts):
        label_name = "Normal" if u == 0 else "Abnormal"
        print(f"   {label_name}: {c}")
    print("-" * 40)
```

```
Subject 118e_6 - Label Distribution:
   Normal: 2166
   Abnormal: 135
----------------------------------------
Subject 118e12 - Label Distribution:
   Normal: 2166
   Abnormal: 135
----------------------------------------
Subject 118e18 - Label Distribution:
   Normal: 2166
   Abnormal: 135
----------------------------------------
Subject 119e_6 - Label Distribution:
   Normal: 1543
```

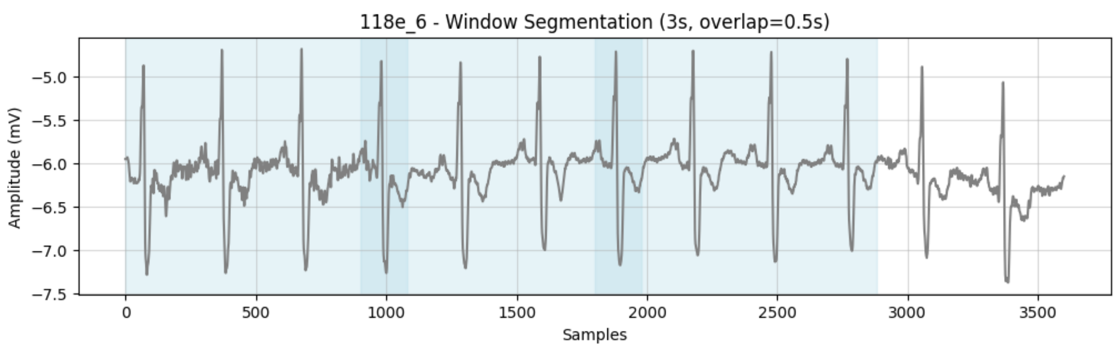
```
   Abnormal: 551
----------------------------------------
Subject 119e24 - Label Distribution:
   Normal: 1543
   Abnormal: 551
----------------------------------------
```

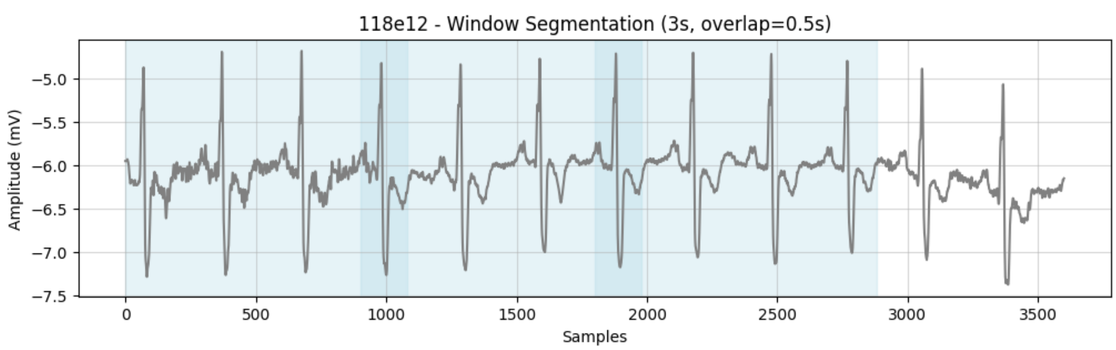
[15]:
```python
def segment_signal(signal, fs, window_size_sec, overlap_sec):
    """
    Splits the ECG signal into overlapping windows.

    Parameters:
        signal (array): ECG signal array
        fs (int): Sampling frequency (Hz)
        window_size_sec (float): window length in seconds
        overlap_sec (float): overlap length in seconds

    Returns:
        windows (np.ndarray): Array of segmented windows
    """
    win_len = int(fs * window_size_sec)
    step = int(fs * (window_size_sec - overlap_sec))
    windows = []
    for start in range(0, len(signal) - win_len, step):
        windows.append(signal[start:start + win_len])
    return np.array(windows)
```

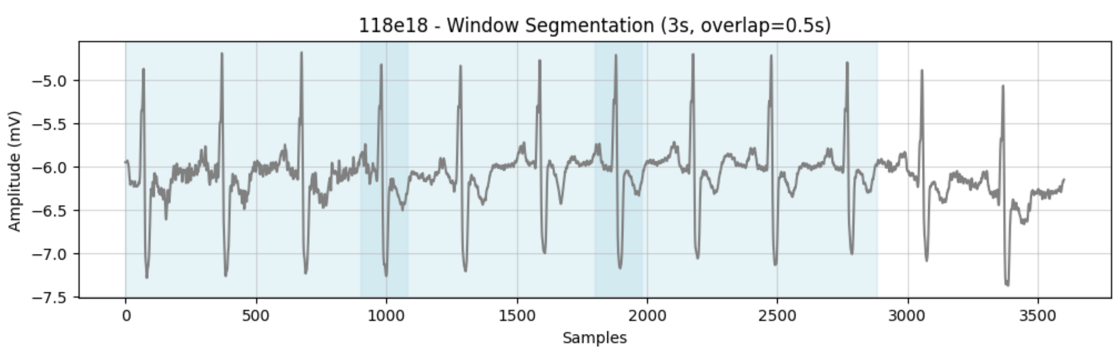
[16]:
```python
fs = 360  # sampling frequency
signal = records['118e_6']['signal']  # example subject
window_sizes = [1, 2, 3, 4, 5]  # in seconds
overlap = 0.5  # in seconds
```

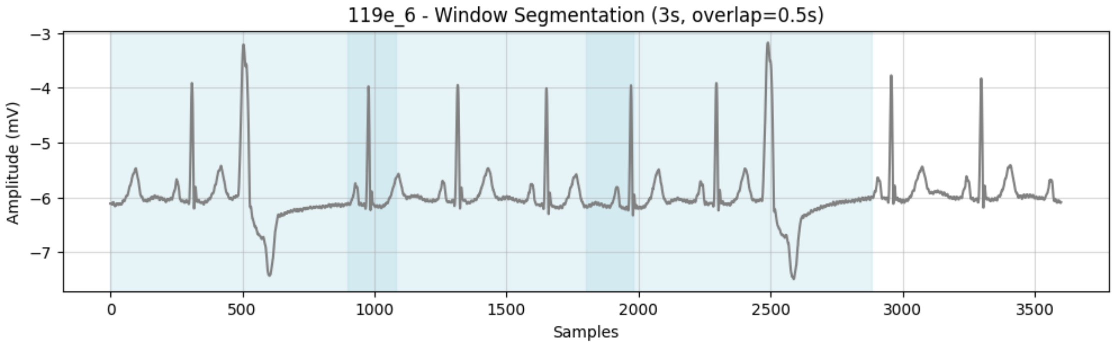
[17]:
```python
def segment_signal(signal, fs, window_size_sec, overlap_sec):
    win_len = int(fs * window_size_sec)
    step = int(fs * (window_size_sec - overlap_sec))
    return np.array([signal[i:i+win_len] for i in range(0, len(signal)-win_len,␣
 ↪step)])

fs = 360
window_size = 3
overlap = 0.5

def plot_windows(signal, fs, ws, overlap, rec_id):
    plt.figure(figsize=(12, 3))
    plt.plot(signal[:fs*10], color='gray')
    step = int(fs * (ws - overlap))
    for i in range(0, int(fs*10 - fs*ws), step):
```

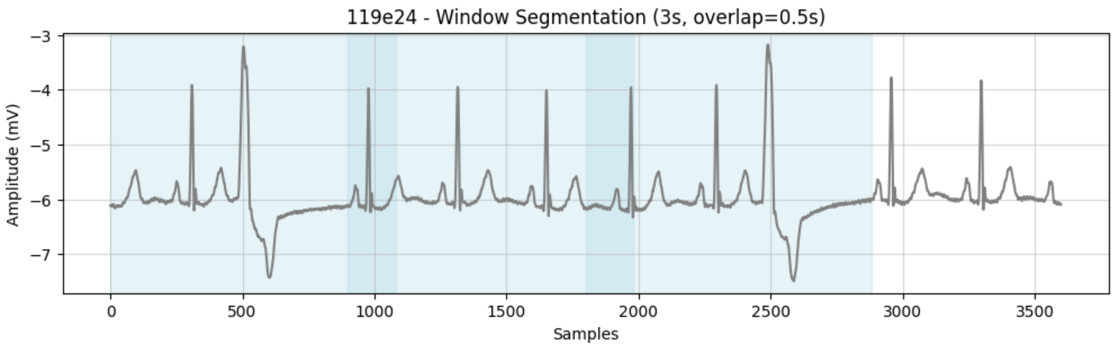
```
        plt.axvspan(i, i+fs*ws, color='lightblue', alpha=0.3)
    plt.title(f'{rec_id} - Window Segmentation ({ws}s, overlap={overlap}s)')
    plt.xlabel('Samples'); plt.ylabel('Amplitude (mV)')
    plt.grid(alpha=0.5); plt.show()

for rec in subjects:
    plot_windows(records[rec]['signal'], records[rec]['fs'], window_size,␣
 ↪overlap, rec)
```











[18]:

```python
# Dictionary to store results: {subject: {window_size: count}}
window_counts_all = {}

for rec in subjects:
    signal = records[rec]['signal']
    counts = {}
    for ws in window_sizes:
        segs = segment_signal(signal, fs, ws, overlap)
        counts[ws] = len(segs)
        print(f"Subject {rec} - Window {ws}s -> {len(segs)} segments")
    window_counts_all[rec] = counts
    print("-" * 50)

# --- Convert to DataFrame for plotting ---
window_df = pd.DataFrame(window_counts_all).T  # rows = subjects, cols = window␣
↪sizes
window_df
```

```
Subject 118e_6 - Window 1s -> 3610 segments
Subject 118e_6 - Window 2s -> 1203 segments
Subject 118e_6 - Window 3s -> 722 segments
Subject 118e_6 - Window 4s -> 515 segments
```

```
Subject 118e_6 - Window 5s -> 401 segments
--------------------------------------------------
Subject 118e12 - Window 1s -> 3610 segments
Subject 118e12 - Window 2s -> 1203 segments
Subject 118e12 - Window 3s -> 722 segments
Subject 118e12 - Window 4s -> 515 segments
Subject 118e12 - Window 5s -> 401 segments
--------------------------------------------------
Subject 118e18 - Window 1s -> 3610 segments
Subject 118e18 - Window 2s -> 1203 segments
Subject 118e18 - Window 3s -> 722 segments
Subject 118e18 - Window 4s -> 515 segments
Subject 118e18 - Window 5s -> 401 segments
--------------------------------------------------
Subject 119e_6 - Window 1s -> 3610 segments
Subject 119e_6 - Window 2s -> 1203 segments
Subject 119e_6 - Window 3s -> 722 segments
Subject 119e_6 - Window 4s -> 515 segments
Subject 119e_6 - Window 5s -> 401 segments
--------------------------------------------------
Subject 119e24 - Window 1s -> 3610 segments
Subject 119e24 - Window 2s -> 1203 segments
Subject 119e24 - Window 3s -> 722 segments
Subject 119e24 - Window 4s -> 515 segments
Subject 119e24 - Window 5s -> 401 segments
--------------------------------------------------
```

[18]:

	1	2	3	4	5
118e_6	3610	1203	722	515	401
118e12	3610	1203	722	515	401
118e18	3610	1203	722	515	401
119e_6	3610	1203	722	515	401
119e24	3610	1203	722	515	401

[19]:

```
results = {}

for rec in ['118e_6', '118e12', '118e18', '119e_6', '119e24']:
    signal = records[rec]['signal']
    segs = segment_signal(signal, fs, window_size_sec=3, overlap_sec=0.5)
    results[rec] = segs
    print(f"Record {rec}: {len(segs)} windows created (3s window, 0.5s␣
↪overlap)")
```

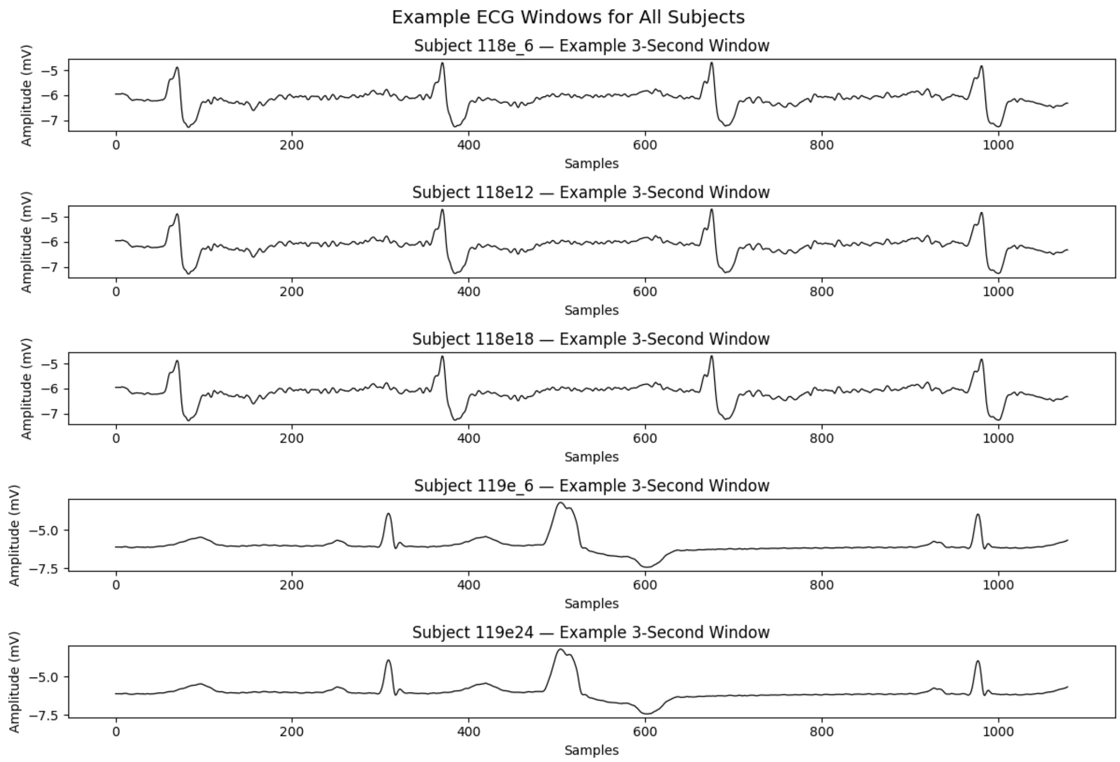
```
Record 118e_6: 722 windows created (3s window, 0.5s overlap)
Record 118e12: 722 windows created (3s window, 0.5s overlap)
Record 118e18: 722 windows created (3s window, 0.5s overlap)
Record 119e_6: 722 windows created (3s window, 0.5s overlap)
Record 119e24: 722 windows created (3s window, 0.5s overlap)
```

[20]:
```python
def stability_analysis(signal, fs, window_lengths, overlaps):
    stability = {}
    for ws in window_lengths:
        for ov in overlaps:
            if ov >= ws:
                # Skip if overlap >= window
                continue
            segments = segment_signal(signal, fs, ws, ov)
            means = [np.mean(s) for s in segments]
            stability[(ws, ov)] = np.std(means)
    return stability

window_lengths = [1, 2, 3, 4, 5]
overlaps = [1, 2, 5, 10]
stability = stability_analysis(signal, fs, window_lengths, overlaps)
# Convert to DataFrame
stab_df = pd.DataFrame(list(stability.items()), columns=['(window, overlap)',␣
 ↪'std_mean'])
stab_df.sort_values(by='std_mean').head()
```

[20]:

	(window, overlap)	std_mean
5	(5, 1)	0.214451
6	(5, 2)	0.214506
4	(4, 2)	0.217772
3	(4, 1)	0.217967
1	(3, 1)	0.221519

[21]:
```python
plt.figure(figsize=(12, 8))

# loop through each subject and plot their first window
for i, rec in enumerate(results.keys(), start=1):
    plt.subplot(len(results), 1, i)  # create one subplot per subject
    plt.plot(results[rec][0], color='black', linewidth=0.9)
    plt.title(f"Subject {rec} - Example 3-Second Window")
    plt.xlabel("Samples")
    plt.ylabel("Amplitude (mV)")
    plt.tight_layout()

plt.suptitle("Example ECG Windows for All Subjects", fontsize=14, y=1.02)
plt.show()
```



[22]:
```
def extract_full_features(windows, fs):
    features = []
    for w in windows:
        # Basic stats
        min_val, max_val = np.min(w), np.max(w)
        amplitude = max_val - min_val
        peaks, _ = find_peaks(w, distance=fs*0.3)
        troughs, _ = find_peaks(-w, distance=fs*0.3)
        rr_interval = np.mean(np.diff(peaks))/fs if len(peaks) > 1 else np.nan
        # Statistical descriptors
        stats = [np.mean(w), np.std(w), np.median(w), skew(w), kurtosis(w)]
        features.append([min_val, max_val, amplitude, len(peaks), len(troughs),␣
 ↪rr_interval] + stats)
    columns =␣
 ↪['Min','Max','Amplitude','Num_Peaks','Num_Troughs','RR_Interval','Mean','Std','Median','Ske
    return pd.DataFrame(features, columns=columns)

combined_features = []
for rec in subjects:
    signal = records[rec]['signal']
    windows = segment_signal(signal, fs, window_size, overlap)
    feat = extract_full_features(windows, fs)
```

```
    feat['Subject_ID'] = rec
    combined_features.append(feat)
combined_features = pd.concat(combined_features, ignore_index=True)
print(f"\nCombined feature shape: {combined_features.shape}")
```

```
Combined feature shape: (3610, 12)
```

[23]:
```
fs = 365               # sampling frequency
window_size = 3        # seconds
overlap = 0.5          # seconds

subjects = ['118e_6', '118e12', '118e18', '119e_6', '119e24']
all_feature_tables = []

for rec in subjects:
    signal = records[rec]['signal']
    windows = segment_signal(signal, fs, window_size, overlap)
    df_feat = extract_full_features(windows, fs)
    df_feat['Subject_ID'] = rec
    all_feature_tables.append(df_feat)
    print(f" Record {rec}: {df_feat.shape[0]} windows × {df_feat.shape[1]}␣
↪features extracted")

# Combine all subjects into one table
combined_features = pd.concat(all_feature_tables, ignore_index=True)
print(f"\nFinal combined feature table shape: {combined_features.shape}")
```

```
 Record 118e_6: 712 windows × 12 features extracted
 Record 118e12: 712 windows × 12 features extracted
 Record 118e18: 712 windows × 12 features extracted
 Record 119e_6: 712 windows × 12 features extracted
 Record 119e24: 712 windows × 12 features extracted

Final combined feature table shape: (3560, 12)
```

[24]:
```
subject_summary = combined_features.groupby("Subject_ID").agg({
    'Min': 'mean',
    'Max': 'mean',
    'Amplitude': 'mean',
    'Num_Peaks': 'mean',
    'Num_Troughs': 'mean',
    'RR_Interval': 'mean',
    'Mean': 'mean',
    'Std': 'mean',
    'Median': 'mean',
    'Skewness': 'mean',
    'Kurtosis': 'mean'
```

```
}).round(4)

# Display neatly
subject_summary = subject_summary.reset_index()
subject_summary
```

[24]:

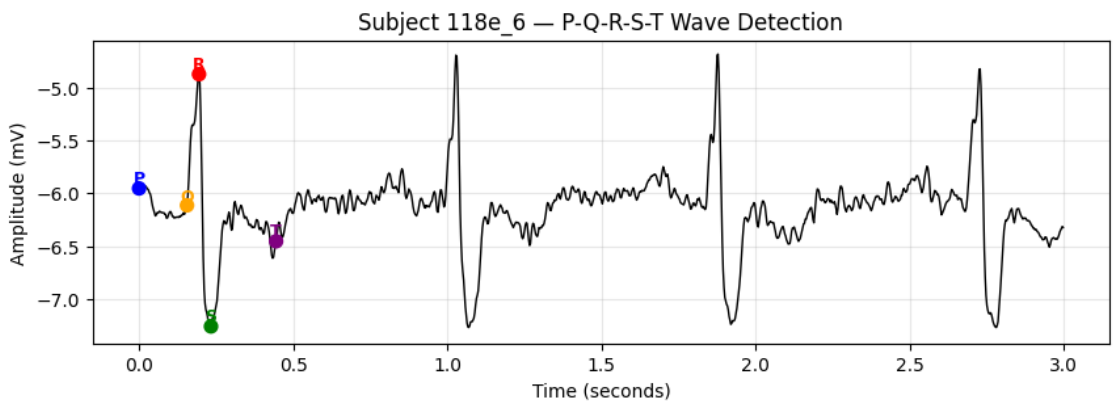
```
  Subject_ID     Min     Max  Amplitude  Num_Peaks  Num_Troughs  RR_Interval  \
0     118e12 -7.1905 -3.8017     3.3888     8.0744       8.0815       0.3912
1     118e18 -7.2253 -4.1231     3.1022     8.0211       8.1067       0.3936
2     118e_6 -7.3338  2.7088    10.0426     8.2486       8.0716       0.3839
3     119e24 -6.9957 -3.2877     3.7080     7.6390       7.9354       0.4164
4     119e_6 -7.1068  2.2534     9.3602     8.0154       7.9958       0.3965

     Mean     Std  Median  Skewness  Kurtosis
0 -5.6871  0.5076 -5.6830   -0.0918    3.0196
1 -5.8295  0.4353 -5.8054   -0.2551    3.5765
2 -3.7329  1.9329 -3.9665    0.1263    2.8280
3 -5.9126  0.5085 -5.9988    2.7574   11.9196
4 -4.0311  1.7696 -4.3016    1.9283    7.3471
```

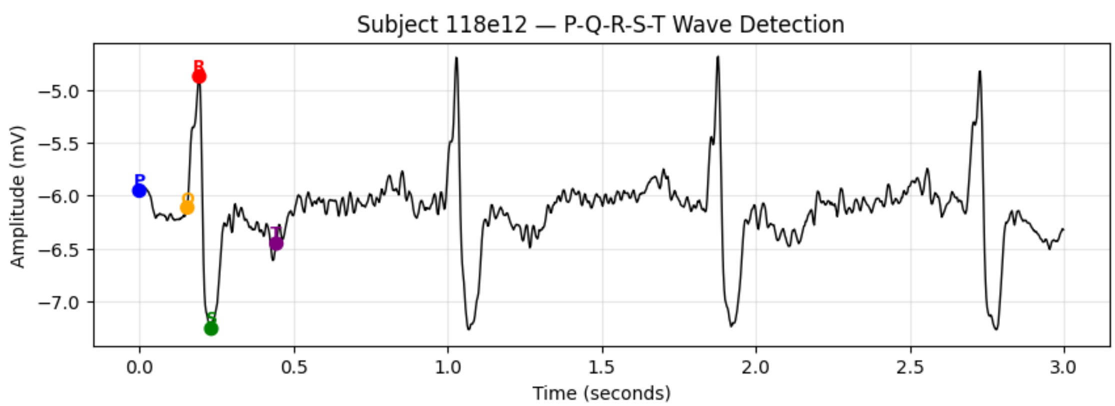
[25]:

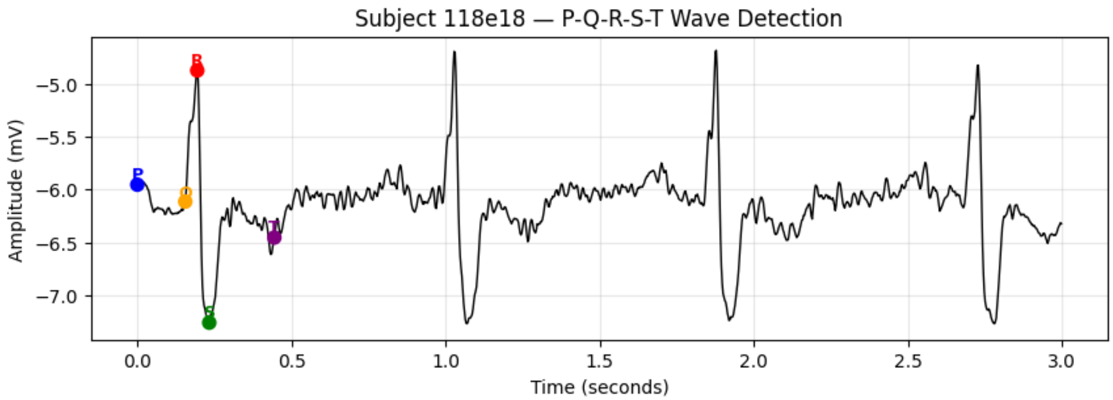
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.signal import find_peaks

fs = 360  # sampling frequency (Hz)

def analyze_and_plot_pqrst(signal, fs, subject_id, duration=3):
    """
    Detects P, Q, R, S, and T waves in an ECG signal segment,
    plots them, and returns amplitude & interval features.
    """
    segment = signal[:fs*duration]
    time_axis = np.arange(len(segment)) / fs

    # Detect R-peaks (dominant spikes)
    r_peaks, _ = find_peaks(segment, distance=fs*0.4, height=np.mean(segment)+0.
 ↪2*np.std(segment))

    # Initialize
    p_amp = q_amp = r_amp = s_amp = t_amp = np.nan
    pr_interval = qrs_duration = qt_interval = np.nan
    p_idx = q_idx = r_idx = s_idx = t_idx = None

    if len(r_peaks) > 0:
        r_idx = r_peaks[0]
        r_amp = segment[r_idx]
```

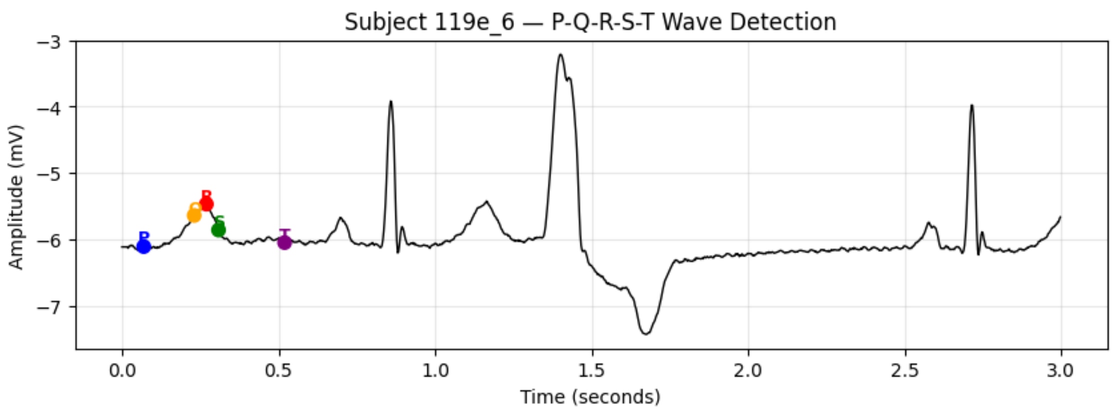
```
        # Approximate other wave positions relative to R
        p_idx = max(0, r_idx - int(0.2 * fs))
        q_idx = max(0, r_idx - int(0.04 * fs))
        s_idx = min(len(segment) - 1, r_idx + int(0.04 * fs))
        t_idx = min(len(segment) - 1, r_idx + int(0.25 * fs))

        p_amp, q_amp, s_amp, t_amp = segment[p_idx], segment[q_idx],␣
↪segment[s_idx], segment[t_idx]

        pr_interval = (r_idx - p_idx) / fs
        qrs_duration = (s_idx - q_idx) / fs
        qt_interval = (t_idx - q_idx) / fs

        # ---- Plot ----
        plt.figure(figsize=(10, 3))
        plt.plot(time_axis, segment, color='black', linewidth=1)
        plt.title(f"Subject {subject_id} - P-Q-R-S-T Wave Detection")
        plt.xlabel("Time (seconds)")
        plt.ylabel("Amplitude (mV)")
        plt.grid(alpha=0.3)

        plt.scatter(time_axis[[p_idx, q_idx, r_idx, s_idx, t_idx]],
                    segment[[p_idx, q_idx, r_idx, s_idx, t_idx]],
                    color=['blue', 'orange', 'red', 'green', 'purple'],
                    s=50, zorder=3)

        labels = ['P', 'Q', 'R', 'S', 'T']
        for idx, label, color in zip([p_idx, q_idx, r_idx, s_idx, t_idx],
                                     labels,
                                     ['blue', 'orange', 'red', 'green',␣
↪'purple']):
            plt.text(time_axis[idx], segment[idx] + 0.05, label,
                     color=color, fontsize=9, ha='center', fontweight='bold')
        plt.show()

    # Return numerical features
    return {
        'Subject_ID': subject_id,
        'P_amp': p_amp, 'Q_amp': q_amp, 'R_amp': r_amp, 'S_amp': s_amp, 'T_amp':
↪ t_amp,
        'PR_interval': pr_interval, 'QRS_duration': qrs_duration, 'QT_interval':
↪ qt_interval
    }


# ---------------------------------------------------
```

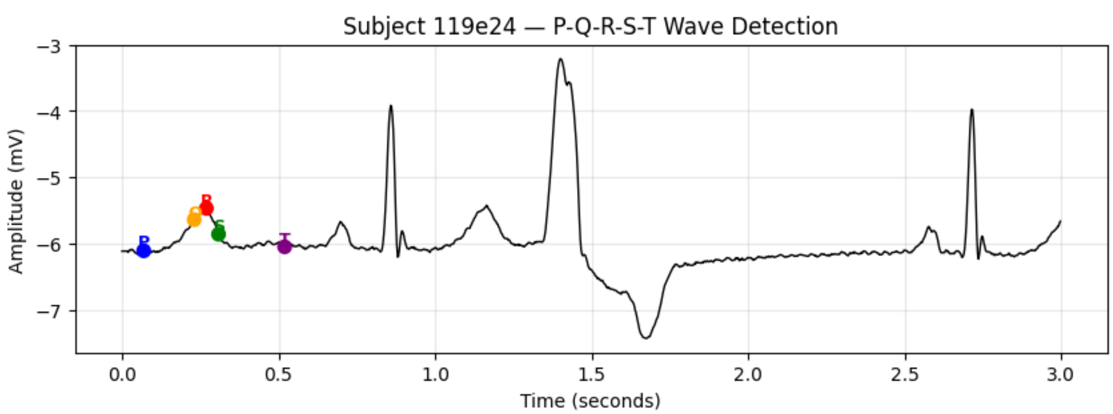
```python
# Run for all five selected subjects and collect data
# --------------------------------------------------
subjects = ['118e_6', '118e12', '118e18', '119e_6', '119e24']
results = []

for rec in subjects:
    signal = records[rec]['signal']
    features = analyze_and_plot_pqrst(signal, fs, rec, duration=3)
    results.append(features)

# Convert to DataFrame for a clean summary
df_pqrst_summary = pd.DataFrame(results)
print("\nExtracted PQRST Amplitude and Interval Features for 5 Subjects:\n")
print(df_pqrst_summary.round(4))
```












```
Extracted PQRST Amplitude and Interval Features for 5 Subjects:
```

```
  Subject_ID  P_amp  Q_amp  R_amp  S_amp  T_amp  PR_interval  QRS_duration  \
0     118e_6 -5.955 -6.115 -4.875  -7.26 -6.455       0.1944        0.0778
1     118e12 -5.955 -6.115 -4.875  -7.26 -6.455       0.1944        0.0778
2     118e18 -5.955 -6.115 -4.875  -7.26 -6.455       0.1944        0.0778
3     119e_6 -6.110 -5.640 -5.470  -5.86 -6.050       0.2000        0.0778
4     119e24 -6.110 -5.640 -5.470  -5.86 -6.050       0.2000        0.0778

   QT_interval
0       0.2889
1       0.2889
2       0.2889
3       0.2889
4       0.2889
```

1 Step 3

[27]:

```python
try:
    df
except NameError:
    df = combined_features.copy()

merged_list = []

for subj in df["Subject_ID"].unique():
    feat_sub = df[df["Subject_ID"] == subj].copy().reset_index(drop=True)
    label_sub = target_df[target_df["Subject_ID"] == subj].copy().
 ↪reset_index(drop=True)

    # Align window counts
    min_len = min(len(feat_sub), len(label_sub))
    feat_sub = feat_sub.iloc[:min_len]
    label_sub = label_sub.iloc[:min_len]

    feat_sub["Target"] = label_sub["Target"]
    merged_list.append(feat_sub)

# Combine all subjects
df_final = pd.concat(merged_list, ignore_index=True)

print(" Final dataset created successfully!")
print("Shape:", df_final.shape)
print(df_final["Target"].value_counts())
```

```
 Final dataset created successfully!
Shape: (3560, 13)
Target
0    2201
```

```
1    1359
Name: count, dtype: int64
```

[28]:
```
X = df_final.drop(columns=["Subject_ID", "Target"])
y = df_final["Target"]

# Balance dataset using SMOTE
sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_resample(X, y)
print("After SMOTE balancing:")
print(pd.Series(y_res).value_counts())

# Split train/test
X_train, X_test, y_train, y_test = train_test_split(
    X_res, y_res, test_size=0.2, random_state=42, stratify=y_res
)

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
print(" Data prepared and scaled.")
```

```
After SMOTE balancing:
Target
1    2201
0    2201
Name: count, dtype: int64
 Data prepared and scaled.
```

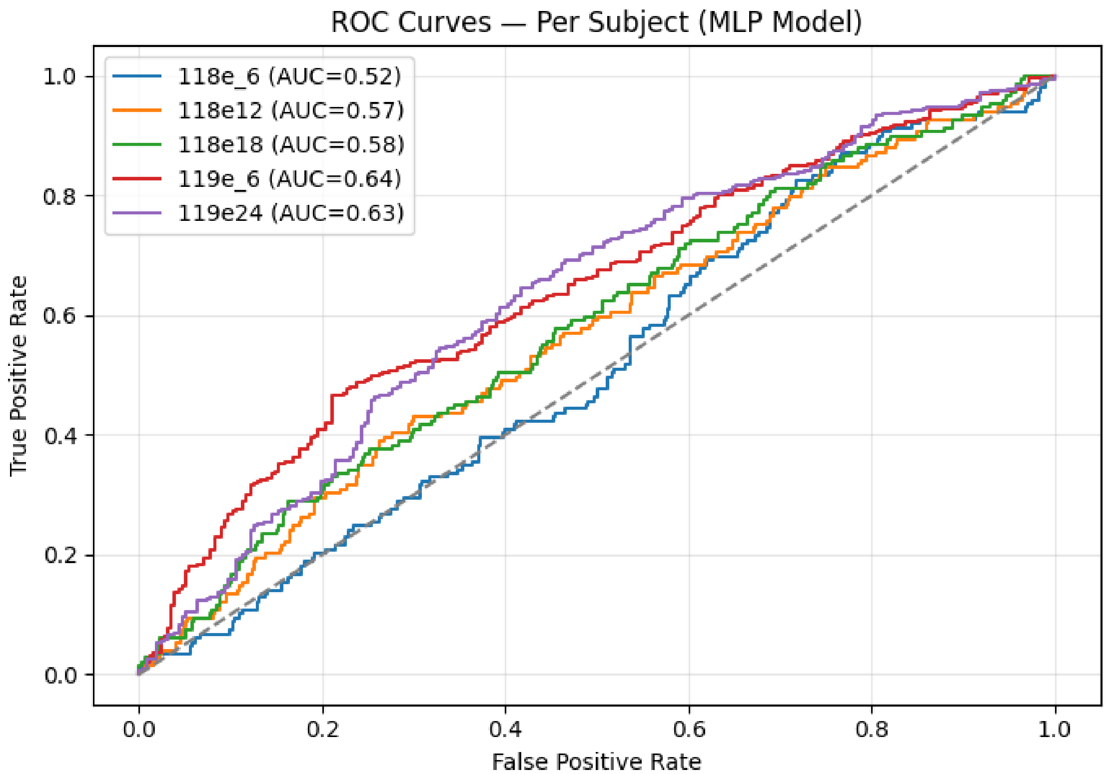
[29]:
```
# Define models
slnn = MLPClassifier(hidden_layer_sizes=(10,), activation='relu',
                     solver='adam', max_iter=500, random_state=42)

mlp = MLPClassifier(hidden_layer_sizes=(32,16), activation='relu',
                    solver='adam', learning_rate_init=0.001,
                    early_stopping=True, max_iter=1000, random_state=42)

# Train models
slnn.fit(X_train_scaled, y_train)
mlp.fit(X_train_scaled, y_train)

print("Models trained successfully!")
```

```
Models trained successfully!
```

[36]:
```
subject_results = []
```

```python
plt.figure(figsize=(7,5))

for subj in subjects:
    # Filter data for this subject and remove missing targets
    sub_df = df_final[df_final['Subject_ID'] == subj].dropna(subset=['Target'])

    if sub_df.empty:
        print(f"Skipping {subj} - no valid Target data.")
        continue

    # Extract features and scale
    X_raw = sub_df.drop(columns=['Subject_ID', 'Target'])
    X_sub = scaler.transform(X_raw)
    y_sub = sub_df['Target'].astype(int)

    # Clean scaled data: handle NaN, inf, and extreme values
    X_sub = np.nan_to_num(X_sub, nan=0.0, posinf=0.0, neginf=0.0)
    imputer = SimpleImputer(strategy='mean')
    X_sub = imputer.fit_transform(X_sub)
    X_sub = np.clip(X_sub, -1e6, 1e6)

    try:
        # Predictions
        y_pred = mlp.predict(X_sub)
        y_prob = mlp.predict_proba(X_sub)[:, 1]

        # Compute metrics
        acc = accuracy_score(y_sub, y_pred)
        f1 = f1_score(y_sub, y_pred)
        auc_val = roc_auc_score(y_sub, y_prob)

        subject_results.append({
            'Subject_ID': subj,
            'Accuracy': round(acc, 3),
            'F1_Score': round(f1, 3),
            'AUC': round(auc_val, 3)
        })

        # Plot ROC Curve for this subject
        fpr, tpr, _ = roc_curve(y_sub, y_prob)
        plt.plot(fpr, tpr, lw=1.5, label=f"{subj} (AUC={auc_val:.2f})")

    except ValueError as e:
        print(f"Skipping {subj} due to error: {e}")

# Combined ROC visualization
plt.plot([0,1], [0,1], '--', color='gray')
```

```
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves - Per Subject (MLP Model)")
plt.legend()
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()

# Display results summary
results_df = pd.DataFrame(subject_results)
print("\n Per-Subject Evaluation Results:\n")
display(results_df)
```



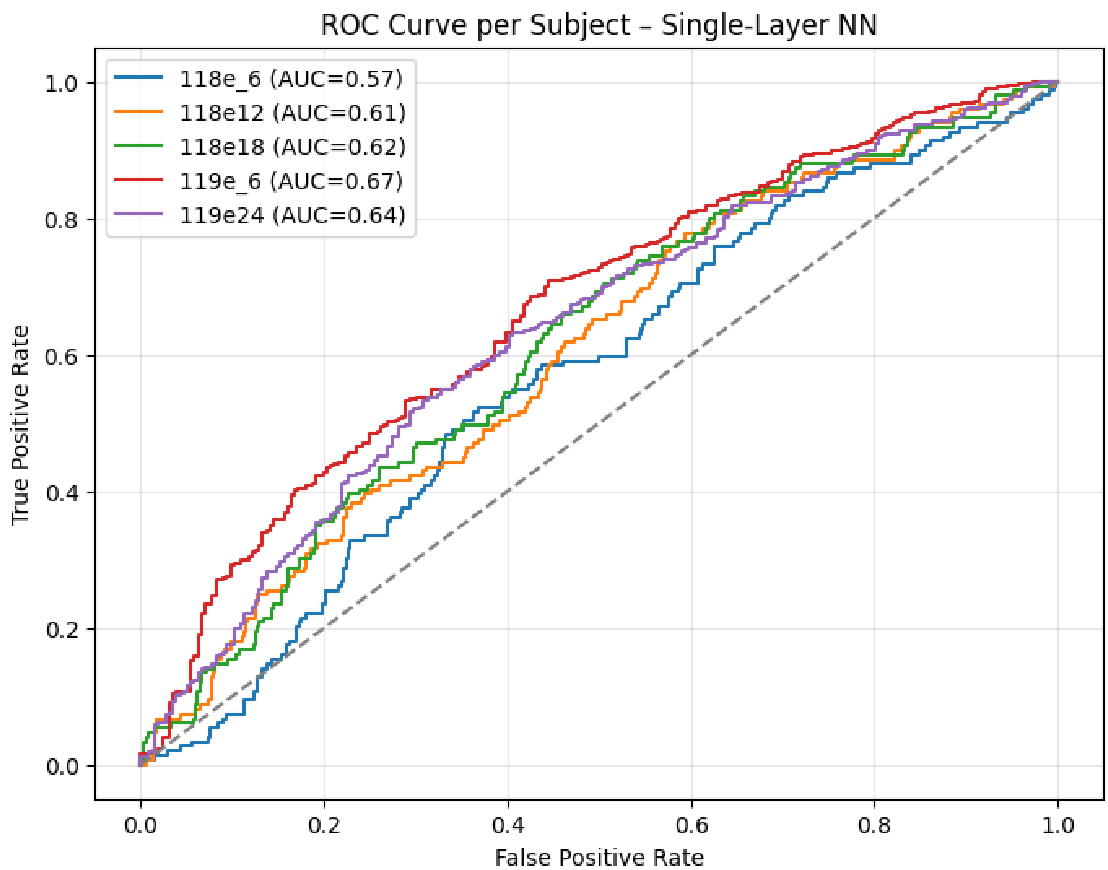

```
 Per-Subject Evaluation Results:

  Subject_ID  Accuracy  F1_Score    AUC
0     118e_6     0.687     0.189  0.520
1     118e12     0.779     0.071  0.567
2     118e18     0.784     0.061  0.580
3     119e_6     0.638     0.734  0.642
4     119e24     0.640     0.779  0.633
```

[30]:
```python
results = []

for name, model in [("Single-Layer NN", slnn), ("Multi-Layer Perceptron", mlp)]:
    y_pred = model.predict(X_test_scaled)
    y_prob = model.predict_proba(X_test_scaled)[:, 1]

    metrics = {
        "Model": name,
        "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),
        "AUC": roc_auc_score(y_test, y_prob)
    }
    results.append(metrics)

    print(f"\n{name} Classification Report:\n", classification_report(y_test,␣
 ↪y_pred))

# Display summary table
results_df = pd.DataFrame(results).round(3)
print("\n=== MODEL COMPARISON ===")
print(results_df)
```

```
Single-Layer NN Classification Report:
               precision    recall  f1-score   support

           0       0.68      0.73      0.70       441
           1       0.71      0.65      0.68       440

    accuracy                           0.69       881
   macro avg       0.69      0.69      0.69       881
weighted avg       0.69      0.69      0.69       881


Multi-Layer Perceptron Classification Report:
               precision    recall  f1-score   support

           0       0.65      0.73      0.69       441
           1       0.69      0.61      0.65       440

    accuracy                           0.67       881
   macro avg       0.67      0.67      0.67       881
weighted avg       0.67      0.67      0.67       881


=== MODEL COMPARISON ===
```

```
                    Model  Accuracy  Precision  Recall     F1    AUC
0          Single-Layer NN     0.691      0.708   0.650  0.678  0.745
1  Multi-Layer Perceptron      0.670      0.692   0.611  0.649  0.725
```

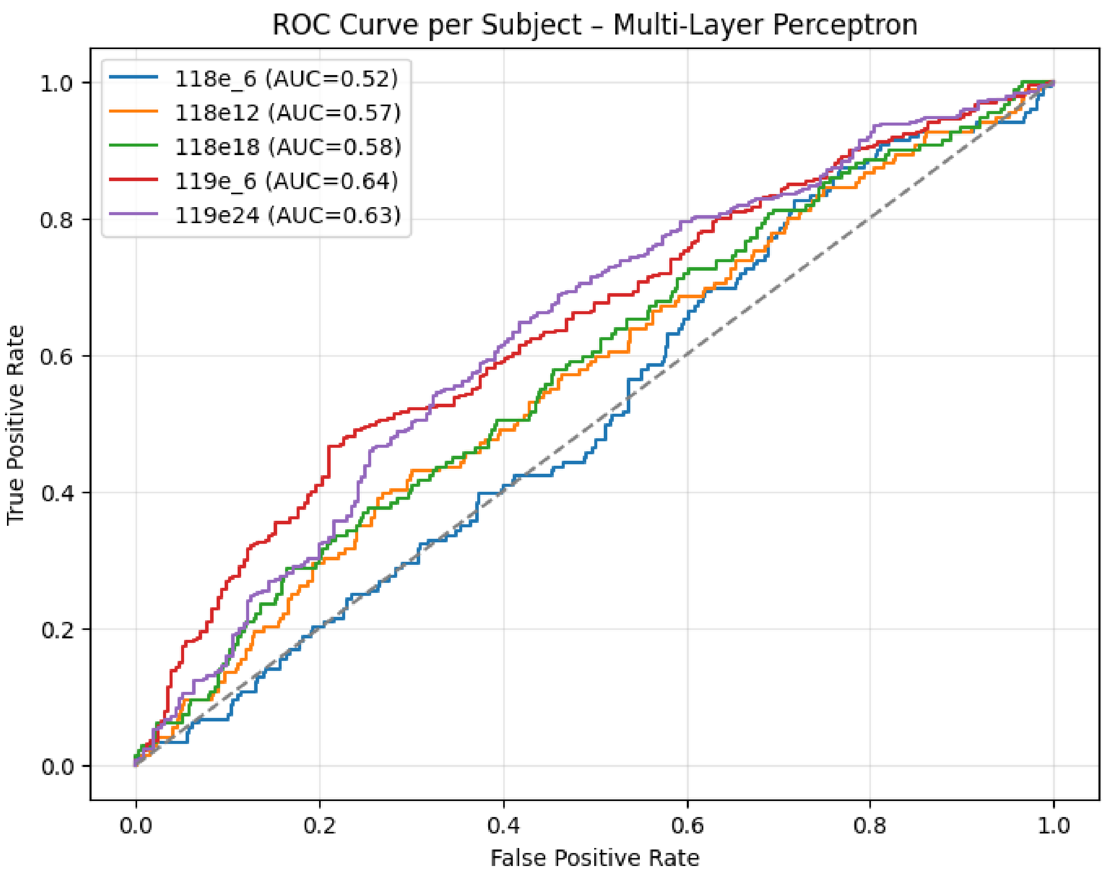
[33]:
```python
subject_results_slnn = []
subject_results_mlp = []

plt.figure(figsize=(8,6))
plt.title("ROC Curve per Subject - Single-Layer NN")
for subj in df_final["Subject_ID"].unique():
    sub_df = df_final[df_final["Subject_ID"] == subj].copy().
 ↪dropna(subset=["Target"])

    # Prepare data
    X_sub = sub_df.drop(columns=["Subject_ID", "Target"])
    y_sub = sub_df["Target"].astype(int)
    X_sub_scaled = scaler.transform(X_sub)

    # Predictions
    y_prob = slnn.predict_proba(X_sub_scaled)[:, 1]
    y_pred = slnn.predict(X_sub_scaled)

    # Metrics
    acc = accuracy_score(y_sub, y_pred)
    f1 = f1_score(y_sub, y_pred)
    auc_val = roc_auc_score(y_sub, y_prob)
    subject_results_slnn.append({"Subject_ID": subj, "Accuracy": acc, "F1": f1,␣
 ↪"AUC": auc_val})

    # ROC curve
    fpr, tpr, _ = roc_curve(y_sub, y_prob)
    plt.plot(fpr, tpr, lw=1.5, label=f"{subj} (AUC={auc_val:.2f})")

plt.plot([0,1],[0,1],'--',color='gray')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.grid(alpha=0.3)
plt.show()

# Convert results to DataFrame
subject_results_slnn = pd.DataFrame(subject_results_slnn)
print("\n=== Subject-wise Performance: Single-Layer NN ===")
print(subject_results_slnn.round(3))
```




```
=== Subject-wise Performance: Single-Layer NN ===
  Subject_ID  Accuracy     F1    AUC
0     118e_6     0.687  0.228  0.569
1     118e12     0.770  0.118  0.606
2     118e18     0.774  0.101  0.619
3     119e_6     0.660  0.758  0.667
4     119e24     0.643  0.778  0.636
```

[34]:
```
plt.figure(figsize=(8,6))
plt.title("ROC Curve per Subject - Multi-Layer Perceptron")
subject_results_mlp = []

for subj in df_final["Subject_ID"].unique():
    sub_df = df_final[df_final["Subject_ID"] == subj].copy().
 ↪dropna(subset=["Target"])

    X_sub = sub_df.drop(columns=["Subject_ID", "Target"])
    y_sub = sub_df["Target"].astype(int)
```

```
    X_sub_scaled = scaler.transform(X_sub)

    y_prob = mlp.predict_proba(X_sub_scaled)[:, 1]
    y_pred = mlp.predict(X_sub_scaled)

    acc = accuracy_score(y_sub, y_pred)
    f1 = f1_score(y_sub, y_pred)
    auc_val = roc_auc_score(y_sub, y_prob)
    subject_results_mlp.append({"Subject_ID": subj, "Accuracy": acc, "F1": f1,␣
 ↪"AUC": auc_val})

    fpr, tpr, _ = roc_curve(y_sub, y_prob)
    plt.plot(fpr, tpr, lw=1.5, label=f"{subj} (AUC={auc_val:.2f})")

plt.plot([0,1],[0,1],'--',color='gray')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.grid(alpha=0.3)
plt.show()

# Convert to DataFrame
subject_results_mlp = pd.DataFrame(subject_results_mlp)
print("\n=== Subject-wise Performance: Multi-Layer Perceptron ===")
print(subject_results_mlp.round(3))
```




```
=== Subject-wise Performance: Multi-Layer Perceptron ===
  Subject_ID  Accuracy     F1    AUC
0     118e_6     0.687  0.189  0.520
1     118e12     0.779  0.071  0.567
2     118e18     0.784  0.061  0.580
3     119e_6     0.638  0.734  0.642
4     119e24     0.640  0.779  0.633
```

2 Optional

[35]:
```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, f1_score, roc_auc_score

# Features and target
X = df.drop(columns=["Subject_ID", "Target"])
y = df["Target"]

# Train-test split
```

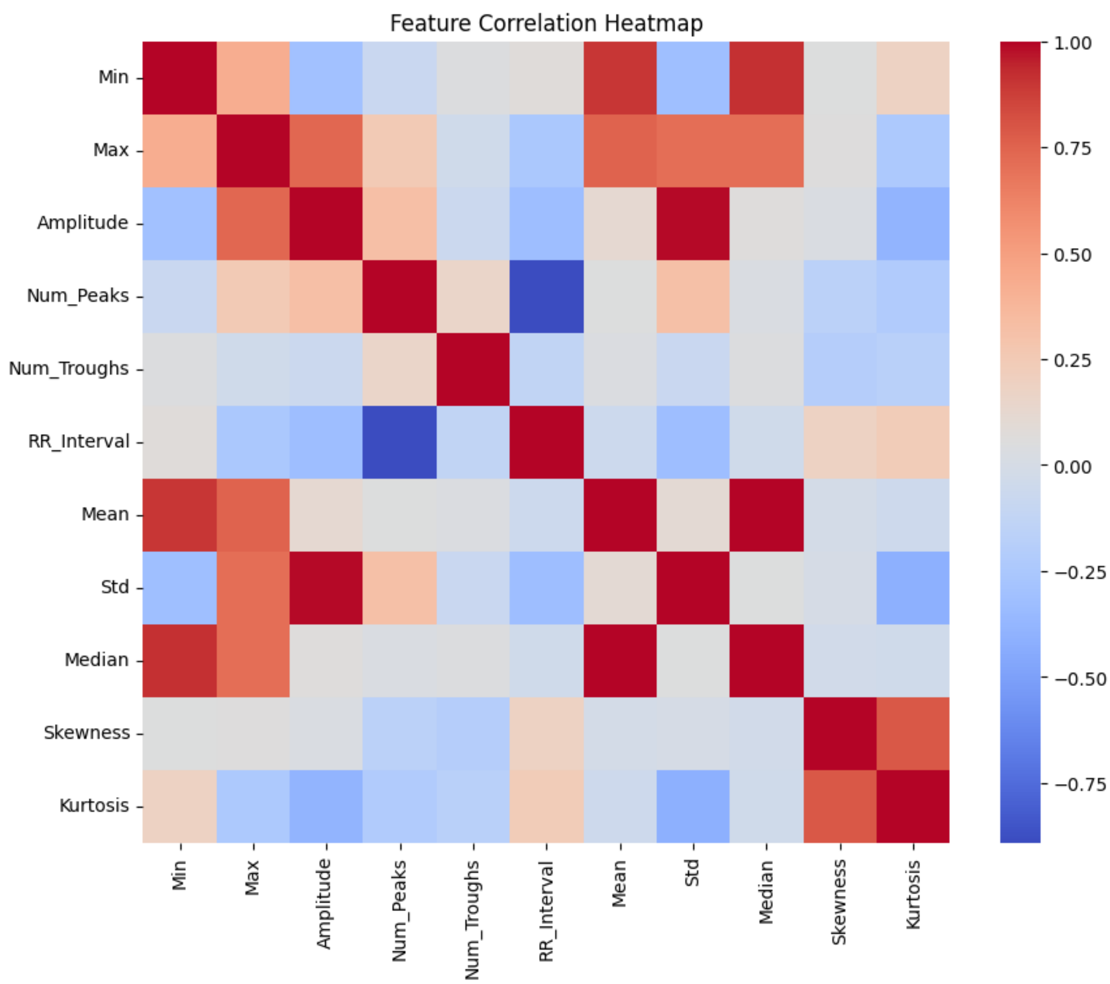
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# Scale
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Single-Layer NN (SLNN)
slnn = MLPClassifier(hidden_layer_sizes=(10,), activation='relu',␣
 ↪solver='adam', max_iter=500, random_state=42)
slnn.fit(X_train_scaled, y_train)
y_pred_slnn = slnn.predict(X_test_scaled)
y_prob_slnn = slnn.predict_proba(X_test_scaled)[:,1]

# Multi-Layer Perceptron (MLP)
mlp = MLPClassifier(hidden_layer_sizes=(32,16), activation='relu',␣
 ↪solver='adam', max_iter=700, random_state=42)
mlp.fit(X_train_scaled, y_train)
y_pred_mlp = mlp.predict(X_test_scaled)
y_prob_mlp = mlp.predict_proba(X_test_scaled)[:,1]

# Compare performance
comparison = pd.DataFrame({
    "Model": ["Single-Layer NN", "Multi-Layer Perceptron"],
    "Accuracy": [accuracy_score(y_test, y_pred_slnn), accuracy_score(y_test,␣
 ↪y_pred_mlp)],
    "F1": [f1_score(y_test, y_pred_slnn), f1_score(y_test, y_pred_mlp)],
    "AUC": [roc_auc_score(y_test, y_prob_slnn), roc_auc_score(y_test,␣
 ↪y_prob_mlp)]
}).round(3)

comparison
```

[35]:

```
                    Model  Accuracy     F1    AUC
0         Single-Layer NN     0.501  0.482  0.475
1  Multi-Layer Perceptron     0.475  0.416  0.473
```

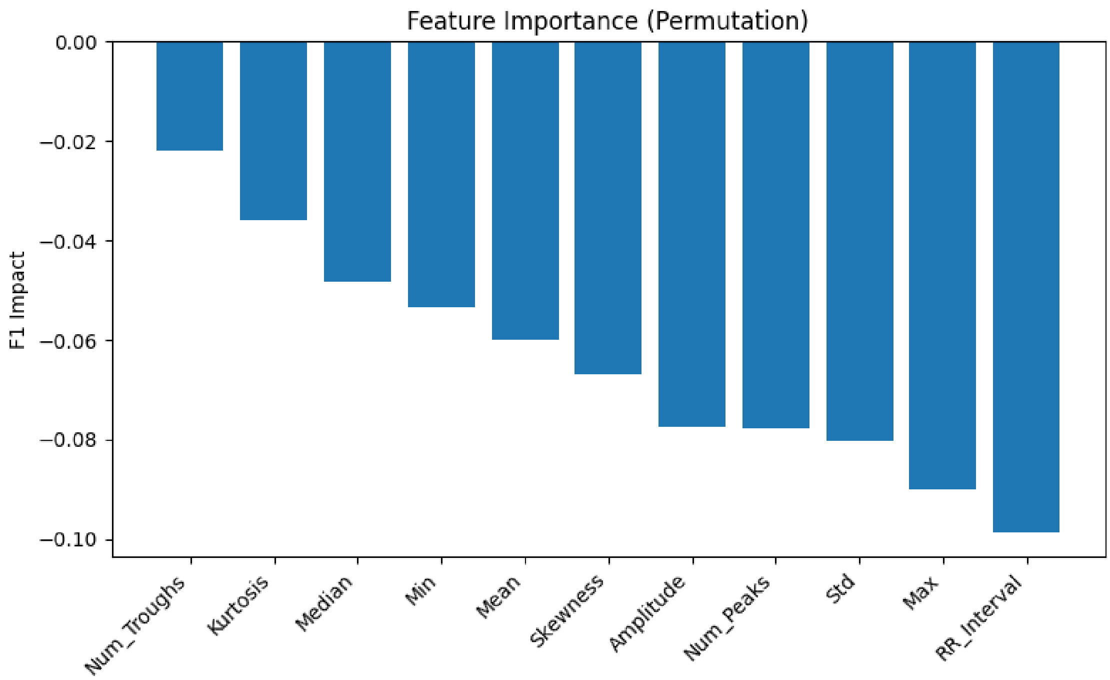
[36]:

```
# Feature Importance and Correlation Visualization
plt.figure(figsize=(10,8))
corr = X.corr()
sns.heatmap(corr, cmap='coolwarm', annot=False)
plt.title("Feature Correlation Heatmap")
plt.show()
```



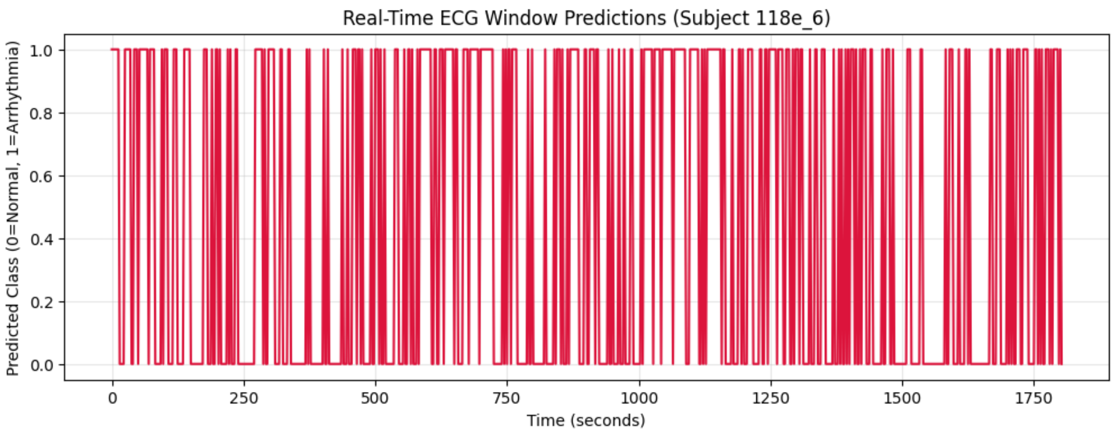
[37]:

```
# Permutation Importance
from sklearn.inspection import permutation_importance

perm_importance = permutation_importance(mlp, X_test_scaled, y_test,␣
↪scoring='f1', n_repeats=10, random_state=42)

sorted_idx = perm_importance.importances_mean.argsort()[::-1]
plt.figure(figsize=(8,5))
plt.bar(range(len(sorted_idx)), perm_importance.importances_mean[sorted_idx])
plt.xticks(range(len(sorted_idx)), X.columns[sorted_idx], rotation=45,␣
↪ha='right')
plt.title("Feature Importance (Permutation)")
plt.ylabel("F1 Impact")
plt.tight_layout()
plt.show()
```



[38]:
```
# Real time window prediction
fs = 360
signal = records['118e_6']['signal']

# Sliding window params
window_size = 3    # seconds
overlap = 0.5      # seconds
win_len = int(fs * window_size)
step = int(fs * (window_size - overlap))

window_preds = []
time_points = []

for start in range(0, len(signal) - win_len, step):
    window = signal[start:start+win_len]
    # extract features same way as before
    feat = extract_full_features([window], fs)
    X_window = scaler.transform(feat[X.columns])  # same scaling as training
    pred = mlp.predict(X_window)[0]
    window_preds.append(pred)
    time_points.append(start / fs)  # seconds

# Convert to DataFrame for plotting
df_real_time = pd.DataFrame({'Time_s': time_points, 'Prediction': window_preds})
```

```
plt.figure(figsize=(12,4))
plt.plot(df_real_time['Time_s'], df_real_time['Prediction'], color='crimson',␣
↪lw=1.5)
plt.title("Real-Time ECG Window Predictions (Subject 118e_6)")
plt.xlabel("Time (seconds)")
plt.ylabel("Predicted Class (0=Normal, 1=Arrhythmia)")
plt.grid(alpha=0.3)
plt.show()
```



[ ]:

```

```