CardioCare AI

Track B — ECG Signal Processing with Python

Reading pack. Nothing to submit. Come with one question.

NumPy · SciPy · NeuroKit2 · ONNX · Python 3.11

See you next week. — Dr. Mahdi Torabi | Torabi Signals Ltd

What Is an ECG Signal?

Your heart produces tiny electrical pulses every time it beats. An ECG machine records those pulses over time. The result is a wave that repeats with every heartbeat. That wave has a tall sharp spike in the middle called the R-peak.

Almost everything in the analysis starts from finding R-peaks. Once you have them, you know exactly when each heartbeat happened. From that, you can calculate heart rate, spot irregular patterns, and measure how much the timing varies between beats.

Sampling rate	How many data points per second. This system uses 128 Hz or 250 Hz.
RR interval	Time in milliseconds between two consecutive R-peaks.
Heart rate	Calculated from RR intervals. 60,000 divided by the average RR interval in ms.
RMSSD	How much consecutive RR intervals differ from each other. A measure of variability.
pNN50	Percentage of consecutive RR pairs where the difference is more than 50ms.

Cleaning the Signal First

Raw ECG signals come with noise baked in. Three types are always present and always need removing before any analysis:

- Baseline wander — slow drift caused by breathing or body movement. Removed with a high-pass filter at 0.5 Hz.
- Powerline hum — constant 50 Hz interference from electrical equipment nearby. Removed with a notch filter.
- High-frequency noise — anything above around 40 Hz. Removed with a low-pass filter.

In Python, using SciPy:

```
from scipy.signal import butter, filtfilt

def bandpass_filter(signal, fs=250):
    nyq = fs / 2
    b, a = butter(4, [0.5 / nyq, 40.0 / nyq], btype='band')
    return filtfilt(b, a, signal)
```

The filtfilt function applies the filter forwards then backwards. This keeps the signal perfectly in time, which matters when you need accurate beat timing.

One thing to watch: the sampling rate fs has to be passed in. If you hardcode 250 but the actual signal was recorded at 128 Hz, every timing calculation will be wrong.

Finding Heartbeats

The Pan-Tompkins algorithm is the standard method for finding R-peaks. It works by differentiating the signal (finding steep slopes), squaring the result (amplifying those slopes), and then looking for peaks above a threshold.

The threshold is adaptive, meaning it adjusts based on the recent history of the signal. A fixed threshold fails when signal amplitude varies across the recording, which is almost always the case in real data.

The codebase uses NeuroKit2 to handle this. NeuroKit2 is a Python library built specifically for physiological signal processing. The main function you'll see is nk.ecg_peaks().

> Critical thing to know:
>
> Some NeuroKit2 functions default to 1000 Hz internally if no sampling rate is passed.
> If your actual signal is at 250 Hz, the detected peaks will be in the wrong positions.
> Always pass sampling_rate explicitly. Every time.

Detecting AFib

Atrial fibrillation is one of the most common serious heart conditions. It causes the gaps between heartbeats to become irregular in a characteristic way. The tricky part is that some irregularity is normal, so you need to measure how irregular.

The detector uses four measurements and combines them with weights:

- Coefficient of variation — standard deviation of RR intervals divided by the mean. Weight: 35%
- RMSSD — short-term beat-to-beat variability. Weight: 30%
- Turning point ratio — how often the RR sequence changes direction. Weight: 20%
- pNN50 — percentage of large consecutive differences. Weight: 15%

```
score = (cov   * 0.35
      + rmssd * 0.30
      + tpr   * 0.20
      + pnn50 * 0.15)

is_afib = score > 0.45
```

If the total is above 0.45, the recording is flagged as AFib. The weighted approach replaced an earlier version that required all four metrics to pass separate hard thresholds. That version was missing too many real cases when one metric was noisy.

> Bug to know about before you start:
>
> The VTVFibDetector class in the codebase doesn't accept a sampling_rate argument,
> even though callers try to pass one. This crashes at runtime with a TypeError.
>
> Your first task will be to fix this. The AFibMLDetector class handles it correctly
> and is the pattern to copy from.

Sliding Windows

Most detection algorithms don't look at the whole recording at once. They use a sliding window: take a chunk of signal, analyse it, move along by a step, take the next chunk, repeat.

For AFib detection, the window is 10 seconds wide and moves 5 seconds at a time. For PVC detection (a different type of abnormal beat), the window is just one beat at a time.

The important thing here is that the window size must be defined in time, not in samples. A 10-second window at 250 Hz is 2,500 samples. At 128 Hz it's 1,280 samples. If you hardcode 2,500 without checking the sampling rate, the algorithm silently breaks on different devices.

```
window_samples = int(window_seconds * sampling_rate)
step_samples   = int(step_seconds   * sampling_rate)
```

That's it. Two lines. But missing that and hardcoding instead is one of the most common bugs in signal processing code.

Before Next Week

Have a look at these. 5 to 10 minutes each is enough:

- NeuroKit2 on GitHub — specifically look at the ecg_peaks() function in the README
- Make sure np.mean, np.std, and np.diff feel familiar
- Have a rough think about what a sliding window means for a long signal

> One question to think about:
>
> If a recording was taken at 128 Hz instead of 250 Hz, what parts of your code would need to change? Just think about it. No need to look anything up.

See you next week. — Dr. Mahdi Torabi | Torabi Signals Ltd