Process a sample photoplethysmogram (PPG) signal using NeuroKit2 library

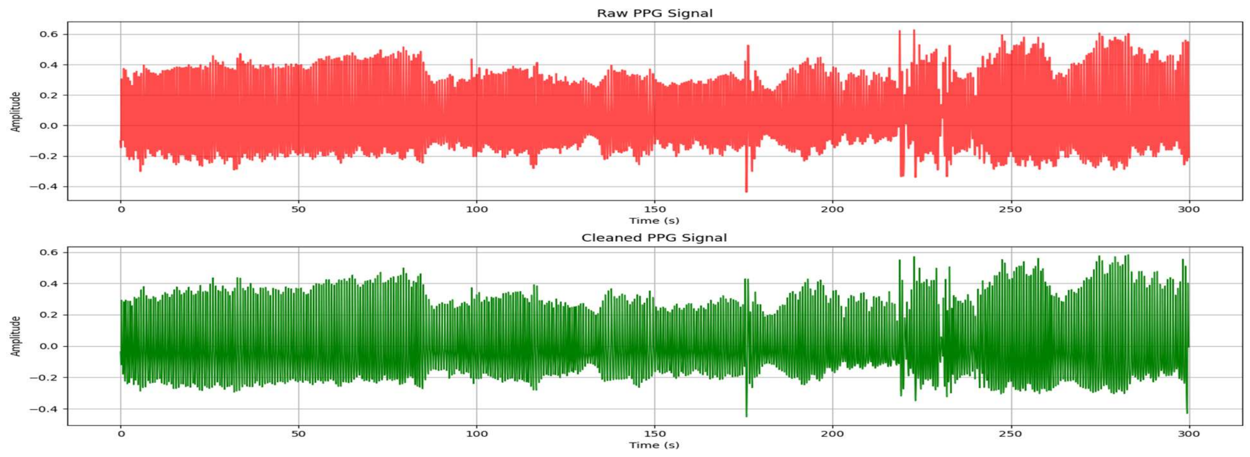
1. Data Loading: It reads a CSV file named "bio_resting_5min_100hz.csv" containing PPG data (ensure it's in the working directory).
2. Signal Processing:

- Extracts the PPG signal from the loaded dataset.
- Defines the sampling rate (fs = 100 samples per second).
- Generates a time vector (time) based on the length of the PPG signal and the sampling rate.

3. Signal Cleaning: Applies a cleaning function (nk.ppg_clean()) from NeuroKit2 to preprocess the PPG signal (ppg_signal). This function typically includes filtering and artifact removal to enhance signal quality.
4. Plotting:
5. Creates a figure with two subplots:
   a. Top subplot (subplot(2, 1, 1)) displays the raw PPG signal over time.
   b. Bottom subplot (subplot(2, 1, 2)) displays the cleaned PPG signal.
6. Visualization:
   • Each subplot is customized with titles, axis labels, grid lines, and color settings to differentiate between the raw and cleaned signals.
   • `plt.tight_layout()` ensures proper spacing between subplots.
7. Each subplot is customized with titles, axis labels, grid lines, and color settings to differentiate between the raw and cleaned signals.
8. plt.tight_layout() ensures proper spacing between subplots.
9. Display: Finally, it shows the plot containing the raw and cleaned PPG signals.

Code:

```
# 1. Load Sample PPG Signal from NeuroKit2
data = pd.read_csv("bio_resting_5min_100hz.csv")  # Ensure it's in the working directory
ppg_signal = data["PPG"]  # Adjust column name if needed
fs = 100   # Sampling rate
time = np.arange(len(ppg_signal)) / fs
ppg_clean = nk.ppg_clean(ppg_signal, sampling_rate=fs)
# Plot raw and cleaned PPG signals in separate subplots
plt.figure(figsize=(14, 8))
# Plot Raw PPG
plt.subplot(2, 1, 1)
plt.plot(time, ppg_signal, color='Red', alpha=0.7)
plt.title("Raw PPG Signal")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.grid(True)
# Plot Cleaned PPG
plt.subplot(2, 1, 2)
plt.plot(time, ppg_clean, color='green', linewidth=1)
plt.title("Cleaned PPG Signal")
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.grid(True)
# Adjust layout and show the plot
plt.tight_layout()
plt.show()
```



1. **Feature Extraction (extract_features function):**
   - Computes key features from the PPG signal, such as:
     - **Heart Rate (BPM)**: Derived from RR intervals between detected peaks.
     - **Pulse Rate Variability (PRV)**: Standard deviation of RR intervals.
     - **Power Spectral Density (PSD)**: Low-frequency (0.04 to 0.15 Hz) and high-frequency (0.15 to 0.4 Hz) power bands.

2.**Dataset Simulation:**

- Generates synthetic PPG signals using NeuroKit2's signal_simulate().
- Extracts features from 500 synthetic signals and assigns random binary labels (0 or 1).

3.**Feature Scaling:**

- Scales features using StandardScaler for better machine-learning performance.
- Splits the data into features (X) and labels (y).

4.**Loading Pre-Trained Model for Prediction:**

- Loads a pre-trained Random Forest classifier (ppg_rf_model.pkl) and a scaler (ppg_scaler.pkl) using joblib.
- Extracts features from a new simulated PPG signal.
- Scales the features using the pre-loaded scaler.

5.**Prediction:**

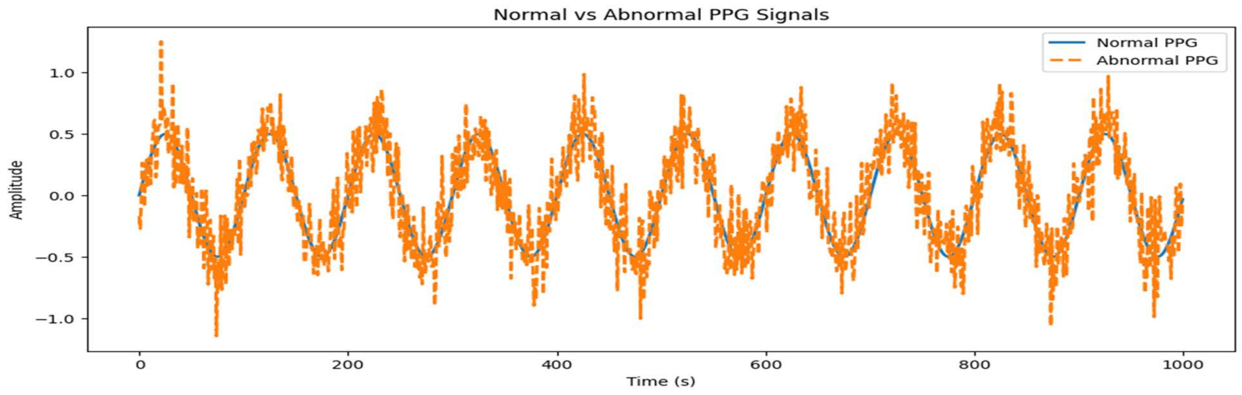
- Makes a prediction on the new sample.
- Outputs whether the PPG signal indicates an **Abnormal** (label 1) or **Normal** (label 0) condition.

Code:

```
# 3. Feature Extraction
def extract_features(ppg_signal, fs):
    features = {}

    # Heart Rate (BPM)
    peaks, _ = signal.find_peaks(ppg_signal, distance=fs*0.6)
    rr_intervals = np.diff(peaks) / fs
    features["Heart_Rate"] = 60 / np.mean(rr_intervals) if len(rr_intervals) > 0 else np.nan
    features["PRV"] = np.std(rr_intervals) if len(rr_intervals) > 0 else np.nan

    # Power Spectral Density (Frequency Features)
    freqs, psd = signal.welch(ppg_signal, fs, nperseg=fs*2)
    features["Low_Freq_Power"] = np.sum(psd[(freqs >= 0.04) & (freqs < 0.15)])
    features["High_Freq_Power"] = np.sum(psd[(freqs >= 0.15) & (freqs < 0.4)])

    return features

features = extract_features(ppg_clean, fs)
features_df = pd.DataFrame([features]).dropna()
print("Extracted Features:")
print(features_df)
# 4. Create Synthetic Dataset for Classification
np.random.seed(42)
df = pd.DataFrame([extract_features(nk.ppg_clean(nk.signal_simulate(duration=10, sampling_rate=fs)),
fs) for _ in range(500)]).dropna()
df['Label'] = np.random.choice([0, 1], size=len(df))  # 0 = Normal, 1 = Abnormal
# 5. Feature Scaling
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X = df.drop(columns=["Label"])
X_scaled = scaler.fit_transform(X)
y = df["Label"]
y
# 9. Load and Test on New Data
clf_loaded = joblib.load("ppg_rf_model.pkl")
scaler_loaded = joblib.load("ppg_scaler.pkl")
```

```
new_sample = extract_features(nk.ppg_clean(nk.signal_simulate(duration=10, sampling_rate=fs)), fs)
new_sample_df = pd.DataFrame([new_sample]).dropna()
new_sample_scaled = scaler_loaded.transform(new_sample_df)

prediction = clf_loaded.predict(new_sample_scaled)
print(f"New Sample Prediction: {'Abnormal' if prediction[0] == 1 else 'Normal'}")
```

1. **Simulate PPG Signals:**
   - A normal PPG signal is generated using NeuroKit2's signal_simulate().
   - An abnormal signal is created by adding random Gaussian noise (np.random.normal(0, 0.2)) to the normal signal to simulate disturbances or abnormalities.
2. **Plotting:**
   - The plot shows both signals on the same graph:
     - **Normal PPG:** Solid line.
     - **Abnormal PPG:** Dashed line.
   - Titles, axis labels, and a legend are added for clarity.



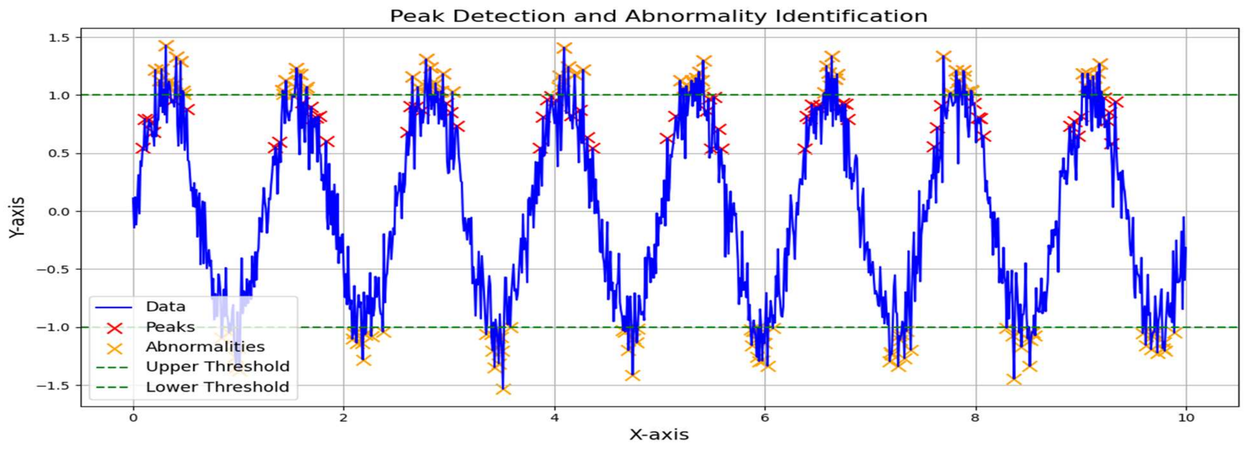
Code:

```
# Plot example of normal and abnormal signals
normal_sample = nk.signal_simulate(duration=10, sampling_rate=fs)
abnormal_sample = normal_sample + np.random.normal(0, 0.2, size=len(normal_sample))

plt.figure(figsize=(12, 5))
plt.plot(normal_sample, label='Normal PPG', linewidth=2)
plt.plot(abnormal_sample, label='Abnormal PPG', linewidth=2, linestyle='dashed')
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.title("Normal vs Abnormal PPG Signals")
plt.legend()
plt.show()
```

1. **Generate Example Data:**

- Creates noisy sinusoidal data using np.sin() and adds Gaussian noise with np.random.normal().

2. **Peak Detection:**
   - Detects peaks using find_peaks(y, height=0.5) where height=0.5 specifies the minimum peak height.
   - Extracts the corresponding y-values for the detected peaks.
3. **Abnormality Identification:**
   - Identifies indices where the signal exceeds the thresholds of 1 or goes below -1 using np.where((y > 1) | (y < -1)).
4. **Visualization:**
   - Plots the main signal, detected peaks (red 'x'), and abnormalities (orange 'x').
   - Adds dashed green lines representing the upper and lower threshold limits.
5. **Peak Count Output:**
   - Prints the total number of detected peaks.



Code:

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

# Generate example data (sinusoidal data with random noise)
x = np.linspace(0, 10, 1000)  # X-axis values
y = np.sin(5 * x) + np.random.normal(0, 0.2, len(x))  # Y-axis values with noise

# Detect peaks using the find_peaks function
# 'height' defines the minimum height of a peak
peaks, properties = find_peaks(y, height=0.5)  # Adjust height as needed
peak_values = y[peaks]  # Get the y-values of the detected peaks
```

```
# Define abnormalities based on thresholds
# Example: Values greater than 1 or less than -1 are marked as abnormalities
abnormal_indices = np.where((y > 1) | (y < -1))[0]  # Indices of abnormalities
abnormal_values = y[abnormal_indices]  # Y-values of abnormalities
print('Total Peak Count ',len(peaks))

# Plot the data
plt.figure(figsize=(12, 6))
plt.plot(x, y, label="Data", color="blue", linewidth=1.5)  # Plot the main data
plt.scatter(x[peaks], peak_values, color="red", marker="x", s=100, label="Peaks")  # Mark peaks
plt.scatter(x[abnormal_indices], abnormal_values, color="orange", marker="x", s=100,
label="Abnormalities")  # Mark abnormalities
plt.axhline(1, color="green", linestyle="--", label="Upper Threshold")  # Upper threshold line
plt.axhline(-1, color="green", linestyle="--", label="Lower Threshold")  # Lower threshold line

# Add plot details
plt.title("Peak Detection and Abnormality Identification", fontsize=16)
plt.xlabel("X-axis", fontsize=14)
plt.ylabel("Y-axis", fontsize=14)
plt.legend(fontsize=12)
plt.grid(True)
plt.tight_layout()

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