```
import numpy as np
import pandas as pd
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
df=pd.read_csv('clean-dataset.csv')
```

```
df.dtypes
```

```
PPG_Signal          int64
Patient_Id          int64
Heart_Rate        float64
Systolic_Peak     float64
Diastolic_Peak    float64
Pulse_Area        float64
index               int64
Gender              int64
Age                 int64
Glucose_level       int64
Height              int64
Weight              int64
pl                  int64
dtype: object
```

```
df
```

```
        PPG_Signal  Patient_Id  Heart_Rate  Systolic_Peak  Diastolic_Peak  \
0              511           1        77.0          522.0           505.0
1              511           1        77.0          522.0           505.0
2              511           1        77.0          522.0           505.0
3              511           1        77.0          522.0           505.0
4              511           1        77.0          522.0           505.0
...            ...         ...         ...            ...             ...
844941         513          23        83.0          516.0           510.0
844942         513          23        83.0          516.0           510.0
844943         513          23        83.0          516.0           510.0
844944         513          23        83.0          516.0           510.0
844945         513          23        83.0          516.0           510.0

        Pulse_Area  index  Gender  Age  Glucose_level  Height  Weight
pl
0            393.0      0       1   38             99     180      53
1
1            393.0      1       1   38            102     180      53
2
2            393.0      2       1   38            103     180      53
3
3            393.0      3       1   38            128     180      53
4
4            393.0      4       1   38            130     180      53
```

```
5
...            ...    ...     ...  ...            ...     ...     ...
...
844941       366.0     43       1   27            108     173      57
1463368
844942       366.0     42       1   27            100     173      57
1463369
844943       366.0     43       1   27            108     173      57
1463370
844944       366.0     42       1   27            100     173      57
1463371
844945       366.0     43       1   27            108     173      57
1463372

[844946 rows x 13 columns]
```

```
missing_values = df.isnull().sum()
print("Missing Values in Each Column:")
print(missing_values)
```

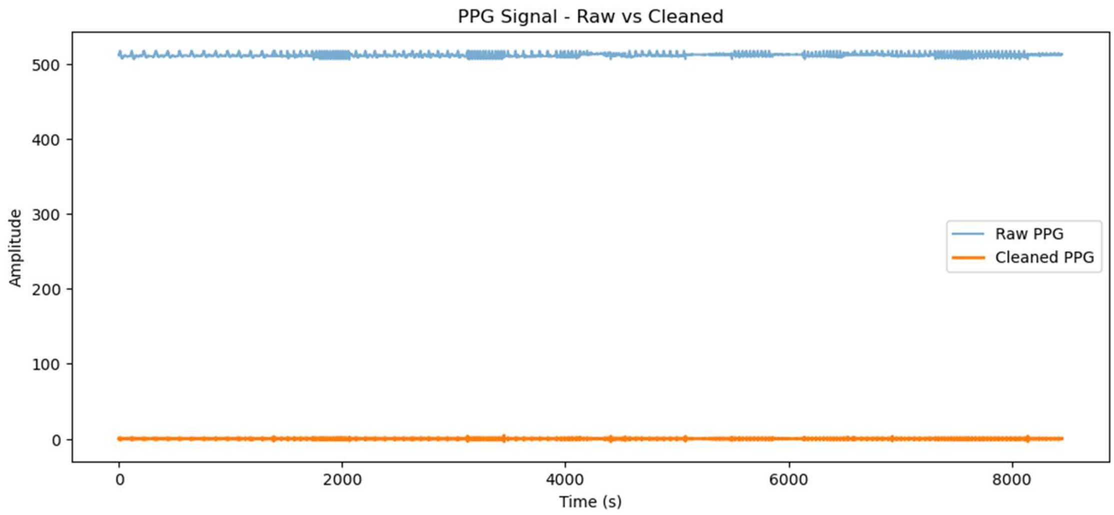
```
Missing Values in Each Column:
PPG_Signal        0
Patient_Id        0
Heart_Rate        0
Systolic_Peak     0
Diastolic_Peak    0
Pulse_Area        0
index             0
Gender            0
Age               0
Glucose_level     0
Height            0
Weight            0
pl                0
dtype: int64
```

```
for column in df.columns:
    unique_values = df[column].unique()
    count_values  = len(df[column].unique())
    print(f"Column: {column}")
    print(f"Unique Values: {unique_values}\n")
    print(f"total count unique values : { count_values}\n")
```

```
Column: PPG_Signal
Unique Values: [511 512 513 514 515 516 517 510 509 508 507 506]

total count unique values : 12

Column: Patient_Id
Unique Values: [ 1  2  3  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21
22 23]

total count unique values : 22

Column: Heart_Rate
Unique Values: [77. 75. 80. 79. 81. 76. 83. 65. 61. 63. 70. 85. 84. 88. 86.
89. 93. 87.
 90. 78. 82. 64. 67. 68. 66. 74. 73. 71. 91. 62. 72. 92.]

total count unique values : 32

Column: Systolic_Peak
Unique Values: [522. 520. 521. 518. 519. 524. 523. 526. 525. 527. 528. 516.
514. 515.
 517. 529.]

total count unique values : 16

Column: Diastolic_Peak
Unique Values: [505. 507. 508. 506. 509. 504. 511. 510. 512.]

total count unique values : 9

Column: Pulse_Area
Unique Values: [393.  406.  383.  385.  386.  375.  394.  380.  376.  396.
399.  369.
 468.  481.5 467.  438.  434.  355.  398.  365.  347.  356.  353.  379.
 345.  363.  367.  364.  378.  312.  370.  349.  377.  366.  321.  324.
 338.  390.  388.  374.  381.  362.  373.  405.  384.  480.  455.  446.
 466.  459.  389.  412.  410.  309.5 400.  477.  465.  354.  417.  433.
 426.  322.  313.  475.  335.  402.  401.  334.  427.  395.  428.  404.
 416.  432.  479.  421.  422.  408.  343.  323.  450.  391.  423.  392.
 342.  431.  333.  346.  357.  403.  424.  397. ]

total count unique values : 92

Column: index
Unique Values: [ 0  1  2  3  4  5  6 31 32 33 44 45 51 52 53 54 55 56 57 58
59 60 61 62
 63 64 65 66  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
 27 28 30 29 34 35 36 38 37 39 40 41 42 43]

total count unique values : 62

Column: Gender
Unique Values: [1 0]

total count unique values : 2
```

```
Column: Age
Unique Values: [38 25 33 23 31 39 37 22 61 50 51 45 24 26 48 27]

total count unique values : 16

Column: Glucose_level
Unique Values: [ 99 102 103 128 130 134 136 108 111 118 120 127  94  96 106
110 129  88
 146 124 100 113  95 115 183 139 112 140]

total count unique values : 28

Column: Height
Unique Values: [180 187 175 165 179 172 182 161 178 157 169 170 154 173]

total count unique values : 14

Column: Weight
Unique Values: [ 53  75 103  56  60  93  63  90  62  61  96  83  89  55  42
88  50  57]

total count unique values : 18

Column: pl
Unique Values: [      1       2       3 ... 1463370 1463371 1463372]

total count unique values : 844946
```

```
ppg_signal = df['PPG_Signal'].values
fs = 100
time = np.arange(len(ppg_signal)) / fs
ppg_clean = nk.ppg_clean(ppg_signal, sampling_rate=fs)
plt.figure(figsize=(12, 5))
plt.plot(time, ppg_signal, label='Raw PPG', alpha=0.6)
plt.plot(time, ppg_clean, label='Cleaned PPG', linewidth=2)
plt.xlabel("Time (s)")
plt.ylabel("Amplitude")
plt.title("PPG Signal - Raw vs Cleaned")
plt.legend()
plt.show()
```



```
# 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)
```

```
Extracted Features:
   Heart_Rate       PRV  Low_Freq_Power  High_Freq_Power
0   40.406097  0.571114             0.0              0.0
```

```
# 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['pl'] = 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=["pl"])
X_scaled = scaler.fit_transform(X)
y = df["pl"]
y
```

```
0      0
1      1
2      0
3      0
4      0
      ..
495    0
496    0
497    1
498    0
499    1
Name: pl, Length: 500, dtype: int32
```

```
# 6. Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)

# 7. Train Model
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)

# # Save the trained model
joblib.dump(clf, "ppg_rf_model.pkl")
joblib.dump(scaler, "ppg_scaler.pkl")
```

```
['ppg_scaler.pkl']
```

```
from scipy.signal import butter, filtfilt, find_peaks

def butter_lowpass_filter(data, cutoff=2.0, fs=100, order=5):
    nyquist = 0.5 * fs
    normal_cutoff = cutoff / nyquist
    b, a = butter(order, normal_cutoff, btype='low', analog=False)
    y = filtfilt(b, a, data)
    return y

# Apply filter to get the cleaned signal
ppg_clean = butter_lowpass_filter(ppg_signal, cutoff=2.0, fs=100, order=5)
```

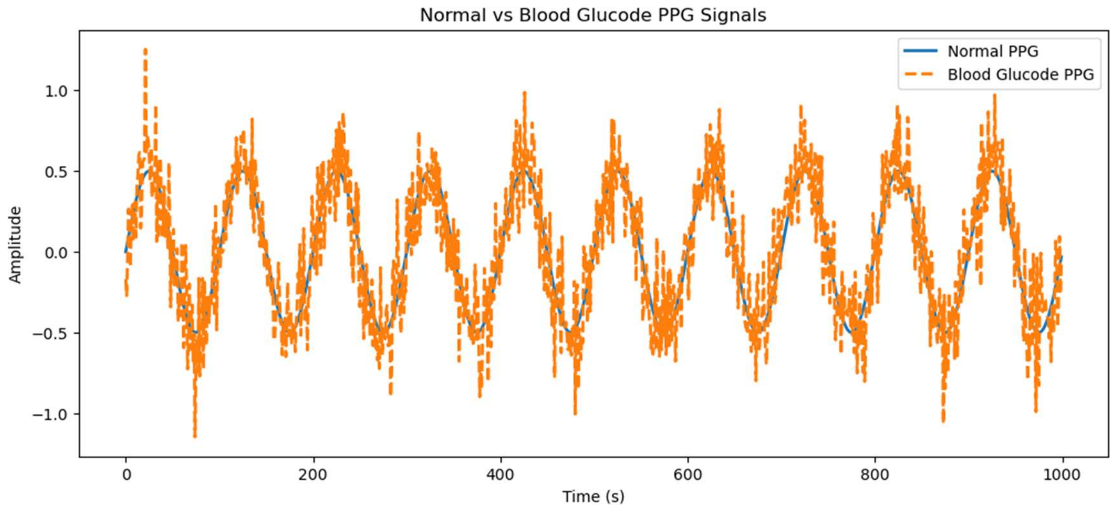
```
# Calculate Signal-to-Noise Ratio (SNR)
def compute_snr(signal, noise):
    power_signal = np.mean(signal**2)
    power_noise = np.mean(noise**2)
    return 10 * np.log10(power_signal / power_noise)

# Compute noise as the difference between raw and cleaned signals
noise = ppg_signal - ppg_clean
snr_value = compute_snr(ppg_clean, noise)

# Calculate RMSE (Lower is better)
rmse_value = np.sqrt(mean_squared_error(ppg_signal, ppg_clean))

# Detect peaks before and after filtering
peaks_raw, _ = find_peaks(ppg_signal, height=np.mean(ppg_signal))
peaks_clean, _ = find_peaks(ppg_clean, height=np.mean(ppg_clean))

# Print evaluation metrics
print(f"Signal-to-Noise Ratio (SNR): {snr_value:.2f} dB")
print(f"Root Mean Square Error (RMSE): {rmse_value:.4f}")
print(f"Number of Peaks - Raw Signal: {len(peaks_raw)}, Cleaned Signal:
{len(peaks_clean)}")
```

```
Signal-to-Noise Ratio (SNR): 73.45 dB
Root Mean Square Error (RMSE): 0.1087
Number of Peaks - Raw Signal: 335, Cleaned Signal: 5347
```

```
# 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: {'Blood Glucose' if prediction[0] == 1 else
'Normal'}")
```

```
New Sample Prediction: Blood Glucose
```

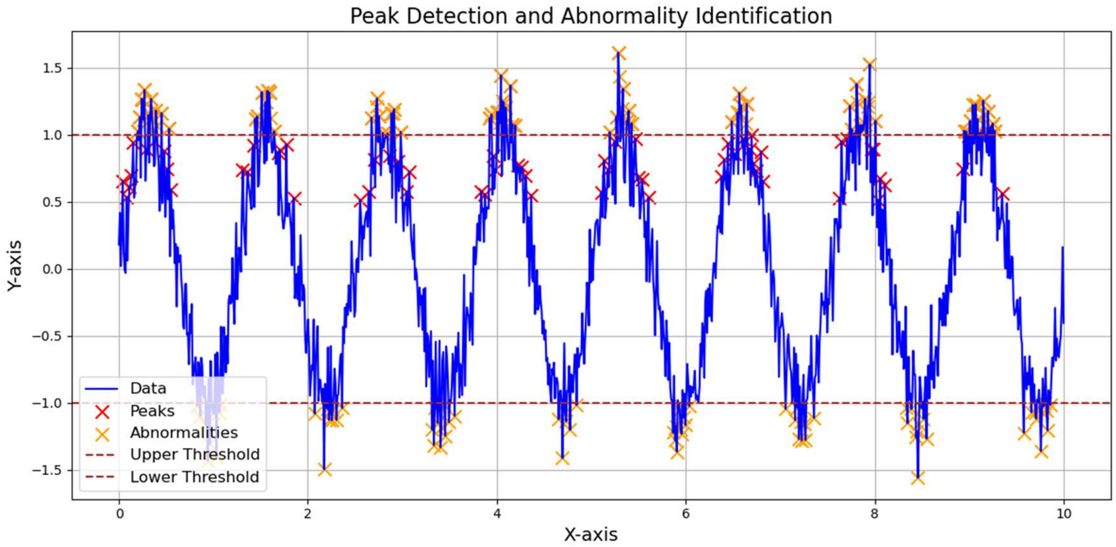
```
# Plot example of normal and abnormal signals
normal_sample = nk.signal_simulate(duration=10, sampling_rate=fs)
abnormal_sample = normal_sample + np.random.normal(0, 0.2,
size=len(normal_sample))

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



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

x = np.linspace(0, 10, 1000)
y = np.sin(5 * x) + np.random.normal(0, 0.2, len(x))

peaks, properties = find_peaks(y, height=0.5)
peak_values = y[peaks]
abnormal_indices = np.where((y > 1) | (y < -1))[0]
abnormal_values = y[abnormal_indices]
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="brown", linestyle="--", label="Upper Threshold")  #
Upper threshold line
plt.axhline(-1, color="brown", 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()
```

```
Total Peak Count  129
```


Peak Detection and Abnormality Identification
Y-axis
X-axis
Data
Peaks
Abnormalities
Upper Threshold
Lower Threshold