ECG Interpretation using Python

1. Importing important libraries:

In [ ]:
```
import os
import pandas as pd
import pandas as pd
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import math
import os
import re
import scipy
from scipy.signal import find_peaks, find_peaks_cwt
```

Reading Data

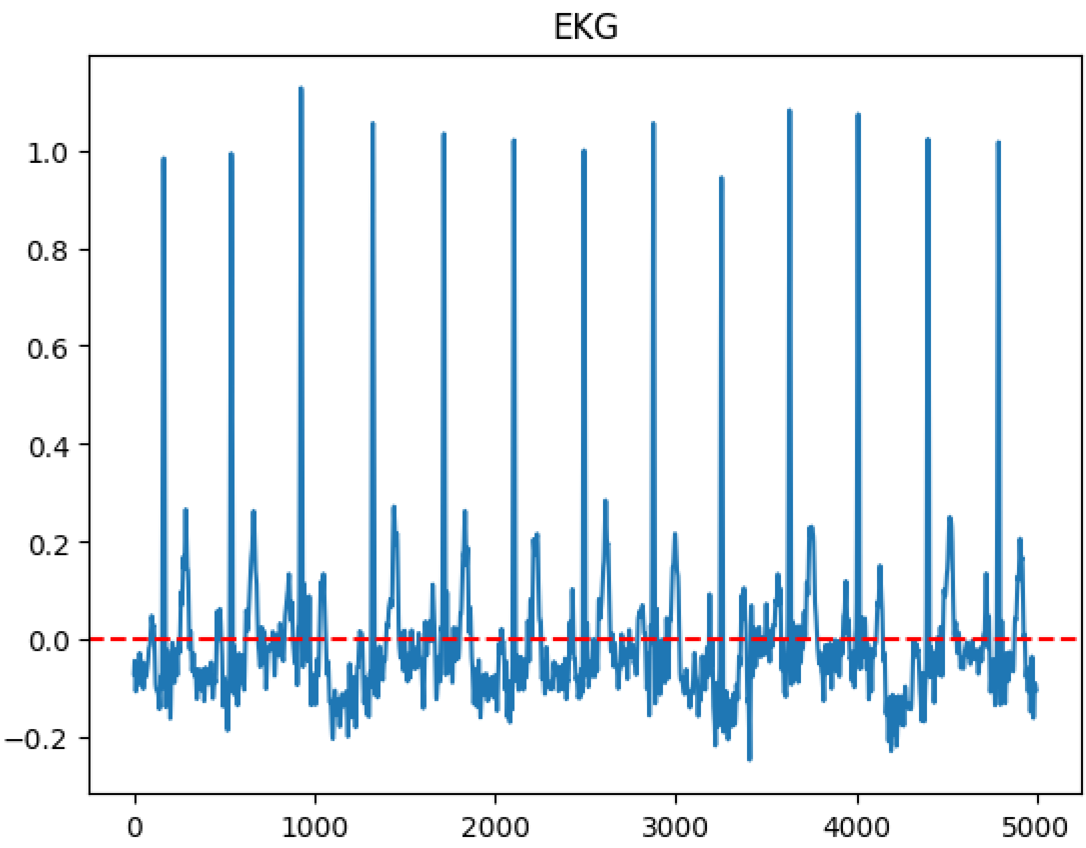
2. Reading the data from the CSV file: This will read the file and return a dataframe.And it also makes sure that file is in CSV or XLSX format.

In [ ]:
```
def read_file(file_path):
    """
    Read a file either as CSV or Excel based on the file extension.

    Parameters:
    - file_path (str): Path to the file.

    Returns:
    - pd.DataFrame: DataFrame containing the data from the file.
    """
    if file_path.endswith('.csv'):
        df = pd.read_csv(file_path)
    elif file_path.endswith('.xlsx') or file_path.endswith('.xls'):
        df = pd.read_excel(file_path)
    else:
        return None

    return df


file_path = r'C:\Users\Divyansh\Desktop\U4RAD\READ_MY_ECG\ALPHA_2023.xlsx'
DF = read_file(file_path)
DF.columns = DF.columns.str.strip()
column_name = 'II'
```

3. Selecting the data from Lead II and plotting the graph.

In [ ]:
```
ekg = DF[column_name].copy()
ekg.plot()
plt.axhline(y=0, color='r', linestyle='--', label="y=0")
```

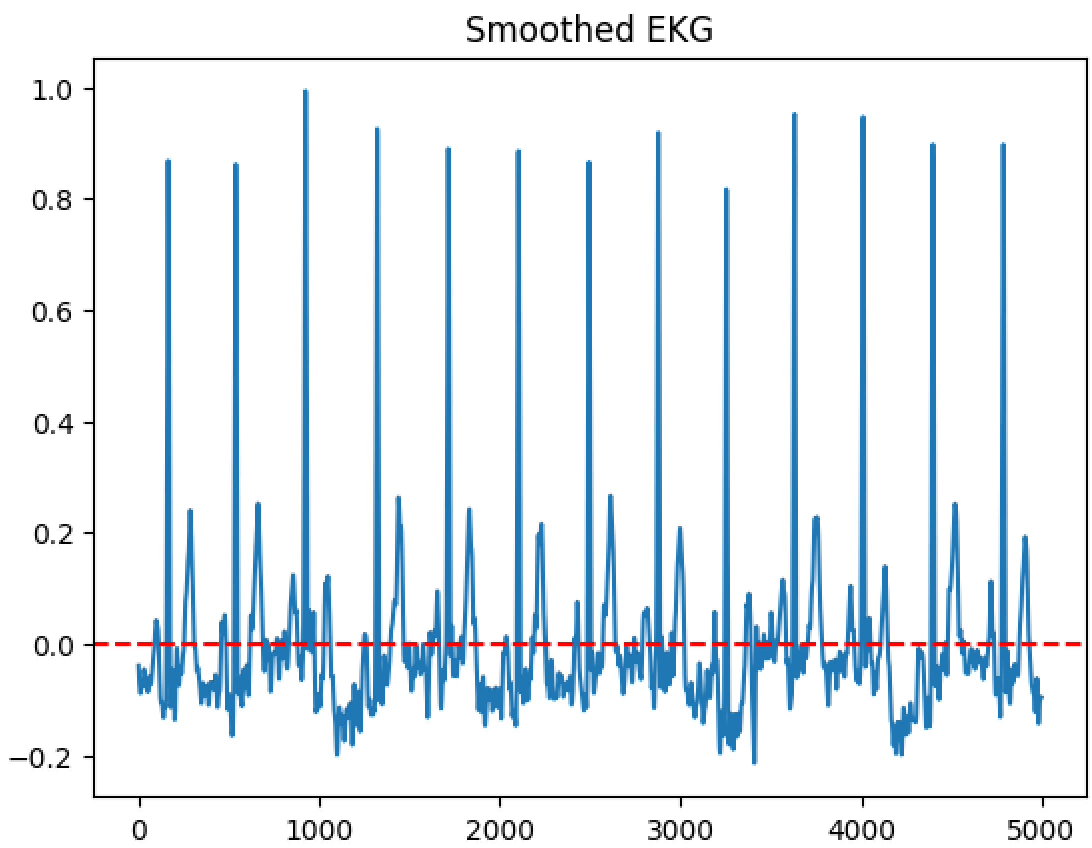
```
plt.title("EKG")
plt.show()
```



Filtering the Data

4. Filtering the data and smoothening it using the Savgol Filter from scipy.

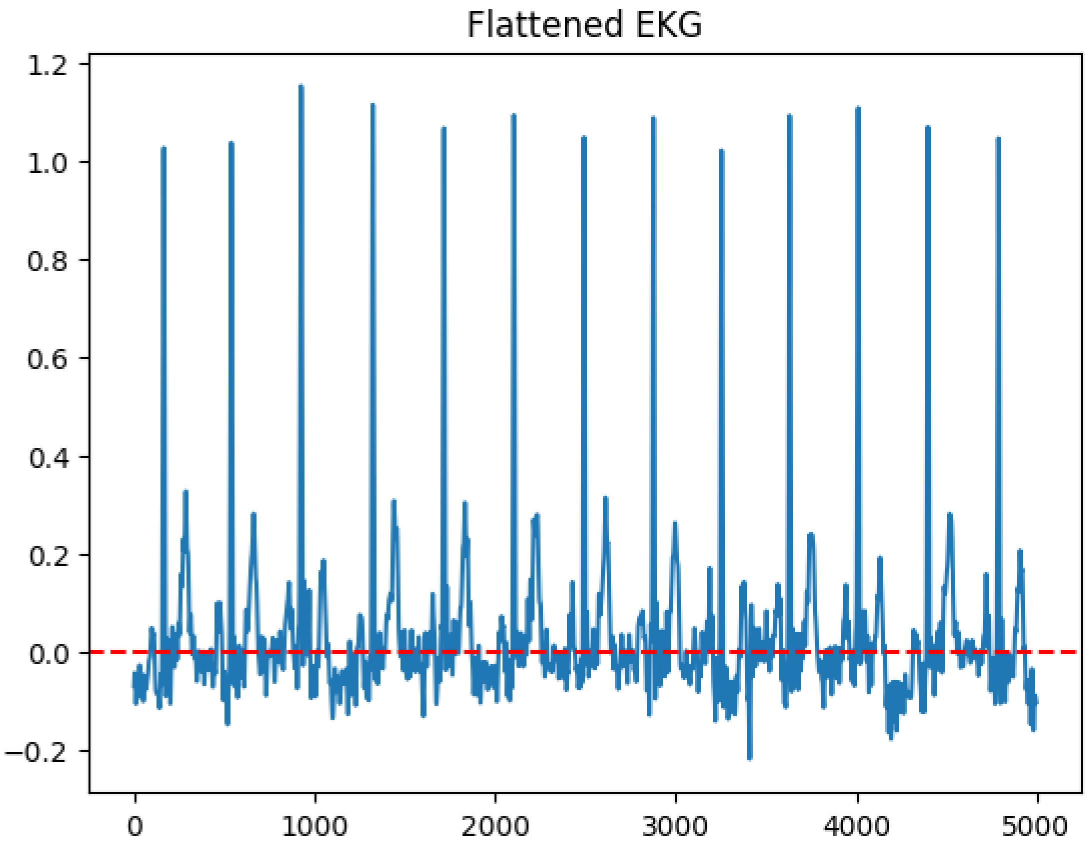
In [ ]:
```
smoothed_heartbeats = scipy.signal.savgol_filter(ekg, window_length=20, polyorde
smoothed_heartbeats = pd.Series(smoothed_heartbeats)
# print(smoothed_heartbeats)
smoothed_heartbeats.plot()
plt.axhline(y=0, color='r', linestyle='--', label="y=0")
plt.title("Smoothed EKG")
plt.show()
```



5. Flattening the graph as to correct the wandering of the graph using a medium filter medfilt() from scipy.

In [ ]:

```
frequency = 500  # Hz,
kernel_size = frequency + 1
wandering_baseline = scipy.signal.medfilt(ekg, kernel_size=kernel_size)
flattened_ekg = ekg - wandering_baseline
flattened_ekg.plot()
plt.axhline(y=0, color='r', linestyle='--', label="y=0")
plt.title("Flattened EKG")
plt.show()
```



RR Interval and Heart Beat

6. Calculating RR Intervals and Heart Beat per minute from the data.

Function to calculate heart beat per minute by taking RR Interval as an arguement.

In [ ]:
```
def heart_beat(interval):
    if(interval==0):
        hbpm = "0"
        return hbpm;
    number_of_blocks = interval/200
    hbpm = 300/number_of_blocks
    return math.floor(hbpm)
```

In [ ]:
```
df = pd.DataFrame({'ecg': flattened_ekg})
column_name = 'ecg'
# Find the R-points.
# Find peaks in the signal
peaks, _ = find_peaks(df[column_name])
# Find troughs in the signal (minima)
troughs, _ = find_peaks(-df[column_name])
try:
      positive_peaks = [peak for peak in peaks if df[column_name][peak] > 0]
      # Filter peaks with values greater than 0.2
      selected_peaks = [peak for peak in peaks if df[column_name][peak] > 0.35]

      # Calculate the average interval between maxima
      if len(selected_peaks) > 1:
         average_interval = sum(selected_peaks[i+1] - selected_peaks[i] for i in
      else:
```

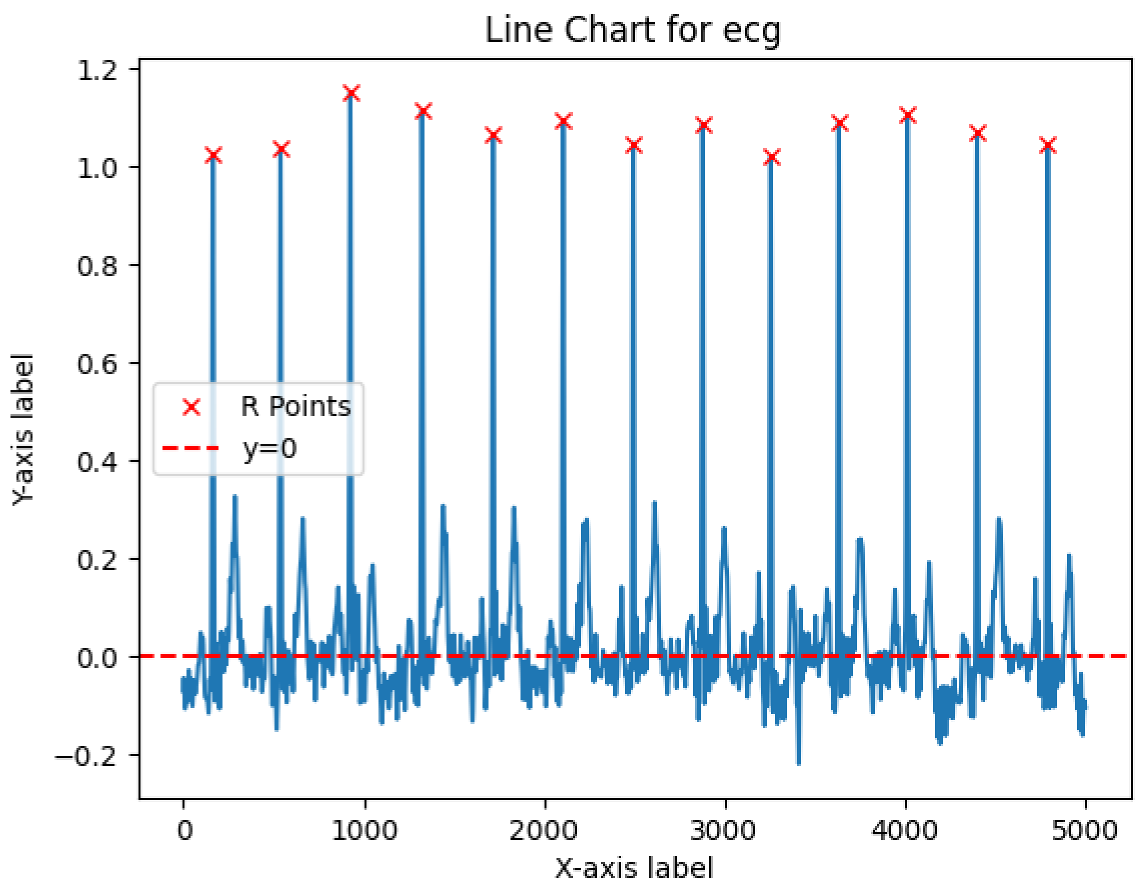
```
            average_interval = 0

        # Calculate the average value of the maxima
        average_maxima_value = sum(df[column_name][peak] for peak in selected_peak

        R_II = "{:.2f}".format(average_maxima_value)
        RR_Interval=math.floor(average_interval)*2
        hbpm = heart_beat(RR_Interval)
except Exception as e:
        print(e)

print("RR Interval:",RR_Interval)
print("R(II):",R_II)
print("HR:",hbpm)
```

```
RR Interval: 770
R(II): 1.07
HR: 77
```

7. Plotting the R-peaks in the graphs.

In [ ]:

```
plt.plot(df[column_name])
plt.plot(selected_peaks, df[column_name][selected_peaks], "x", label="R Points",
plt.axhline(y=0, color='r', linestyle='--', label="y=0")
# Adding labels and title
plt.xlabel('X-axis label')
plt.ylabel('Y-axis label')
plt.title('Line Chart for {}'.format(column_name))
# Display the chart
plt.legend()
plt.show()
```



QRS Complex

8. Recognizing the QRS complex and finding out QRS duration.

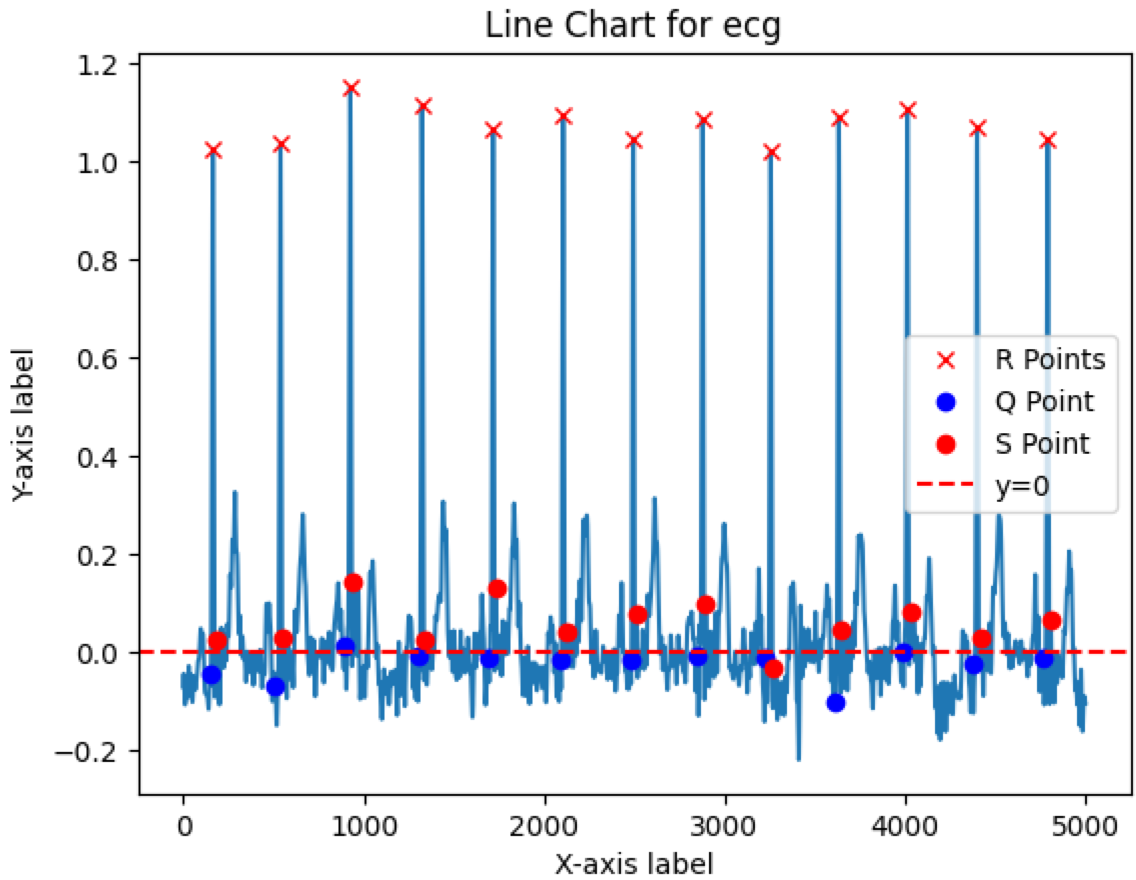
In [ ]:
```
#QRS
try:
    # Initialize lists to store Q and S points
    q_points = []
    s_points = []
    # Identify Q and S points for each R point

    for r_peak in selected_peaks:
        # Find troughs before and after the R peak
        troughs_before_r = [trough for trough in troughs if trough < r_peak]
        troughs_after_r = [trough for trough in troughs if trough > r_peak]

        # Find the Q point (minima with negative values just before R)
        q_point = max((trough for trough in troughs_before_r if df[column_name][

        # Find the S point (minima with negative values just after R)
        s_point = min((trough for trough in troughs_after_r if df[column_name][t

        # Append Q and S points to the respective lists
        q_points.append(q_point)
        s_points.append(s_point)

    newQ_points = []
    for point in q_points:
        # Find the peak just to the left of the point
        newQ_point = max((peak for peak in peaks if peak < point), default=None)
```

```
        # Append the found peak to the list
        newQ_points.append(newQ_point)
    newS_points = []
    for point in s_points:
        # Find the peak just to the left of the point
        newS_point =  min((peak for peak in peaks if peak > point), default=None
        # Append the found peak to the list
        newS_points.append(newS_point)


    QRS_durations = []
    for q_point, s_point in zip(newQ_points, newS_points):
        if q_point is not None and s_point is not None:
            duration = s_point - q_point
            QRS_durations.append(duration)
    # Calculate average duration
    average_duration_qrs = sum(QRS_durations) / len(QRS_durations) if len(QRS_du
    QRS_avg = math.floor(average_duration_qrs*2)
except Exception as e:
    print(e)
    QRS_avg=0
    # print("QRS BLOCK",patient_name,e)
print("QRS:",QRS_avg)
```

```
QRS: 81
```

9. Plotting the QRS Complex.

In [ ]:
```
plt.plot(df[column_name])
plt.plot(selected_peaks, df[column_name][selected_peaks], "x", label="R Points",
plt.plot(newQ_points, df[column_name][newQ_points], "o", label="Q Point", color=
plt.plot(newS_points, df[column_name][newS_points], "o", label="S Point", color=
plt.axhline(y=0, color='r', linestyle='--', label="y=0")
# Adding labels and title
plt.xlabel('X-axis label')
plt.ylabel('Y-axis label')
plt.title('Line Chart for {}'.format(column_name))
# Display the chart
plt.legend()
plt.show()
```



QT Interval

10. Finding T-peaks and calculating values of QT, QTc and QT/QTc Ratio.

In [ ]:
```
def find_maximum_between_ranges(r_point, p_point):
    max_values = []

    for i in range(len(r_point) - 1):
        start_index = p_point.index(r_point[i])
        end_index = p_point.index(r_point[i + 1])
        values_within_range = p_point[start_index + 1:end_index]

        if values_within_range:
            max_value = max(values_within_range)
            max_values.append(max_value)
        else:
            max_values.append(None)

    return max_values
```

In [ ]:
```
r_points = []
for r_point in selected_peaks:
    r_points.append(df[column_name][r_point])
r_pair = [(key,value) for i, (key,value) in enumerate(zip(selected_peaks, r_poin
r_dict = dict(r_pair)
#print(r_dict)

p_points = []
for p_point in positive_peaks:
```

```
    p_points.append(df[column_name][p_point])
# print("P-points",p_points)
p_pair = [(key,value) for i, (key,value) in enumerate(zip(positive_peaks, p_poin
p_dict = dict(p_pair)
# print("P DICT:",p_dict)
```

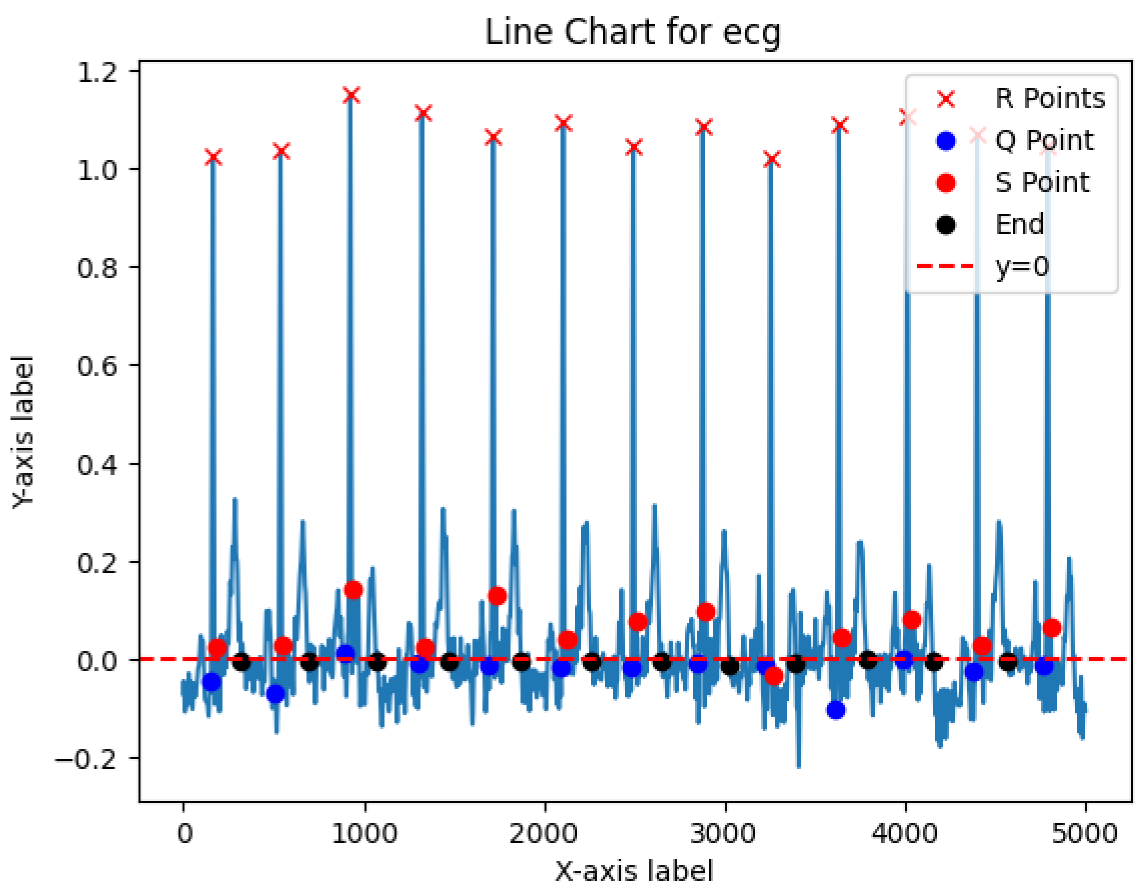
In [ ]:
```
t_point = find_maximum_between_ranges(r_points,p_points)
# Create a dictionary where keys are taken from p_dict and values from t_point
t_dict = {key: value for key, value in p_dict.items() if p_dict[key] in t_point}
#print("T Dictionary:", t_dict)

t_keys_list = [key for key in t_dict]
#print("T Keys List:", t_keys_list)
wave_end=[]
for key in t_keys_list:
    for i in range(key,len(df.values)):
        if (df[column_name][i]<0):
            wave_end.append(i)
            break;
    # qt_durations = [t - q for q, t in zip(q_points, wave_end)]
QT_durations = []
for q_point, t_point in zip(newQ_points, wave_end):
    if q_point is not None and t_point is not None:
        duration = t_point - q_point
        QT_durations.append(duration)
avg_duration = (sum(QT_durations)/len(QT_durations))*2
QT_avg = math.floor(avg_duration)
QT_max = max(QT_durations)*2
QT_min = min(QT_durations)*2
if QT_avg>500:
    QT_avg = QT_min
# print(QT_durations)
# QT_interval = max(qt_durations)*2
QTC =  math.ceil(QT_avg/ math.sqrt(60/hbpm))
QT_QC_Ratio = "{:.2f}".format(QT_avg/QTC)

print("QT:",QT_avg)
print("QTc:",QTC)
print("QT/QTc:",QT_QC_Ratio)
```

```
QT: 347
QTc: 394
QT/QTc: 0.88
```

11. Plotting the wave end of the heartbeat.

In [ ]:
```
plt.plot(df[column_name])
plt.plot(selected_peaks, df[column_name][selected_peaks], "x", label="R Points",
plt.plot(newQ_points, df[column_name][newQ_points], "o", label="Q Point", color=
plt.plot(newS_points, df[column_name][newS_points], "o", label="S Point", color=
plt.plot(wave_end, df[column_name][wave_end], "o", label="End", color='black')
plt.axhline(y=0, color='r', linestyle='--', label="y=0")
# Adding labels and title
plt.xlabel('X-axis label')
plt.ylabel('Y-axis label')
plt.title('Line Chart for {}'.format(column_name))
# Display the chart
```

```
plt.legend()
plt.show()
```



PR Interval

12. Finding the P-points and Calculating PR Interval.

In [ ]:
```
def find_max_peak_in_range(df, column_name, start_index, end_index):
        # Ensure that the range is valid
        if start_index < 0 or end_index >= len(df):
            print("Invalid range.")
            return None

        # Extract the specified range from the DataFrame column
        range_data = df[column_name][start_index:end_index+1]
        # Find peaks within the range
        peaks = [(peak, start_index + i) for i, peak in enumerate(range_data)]
        if not peaks:
            return None  # No peaks found, return None
        else:
            # Find the maximum peak within the range
            max_peak = max(peaks, key=lambda x: x[0])

        return max_peak
```

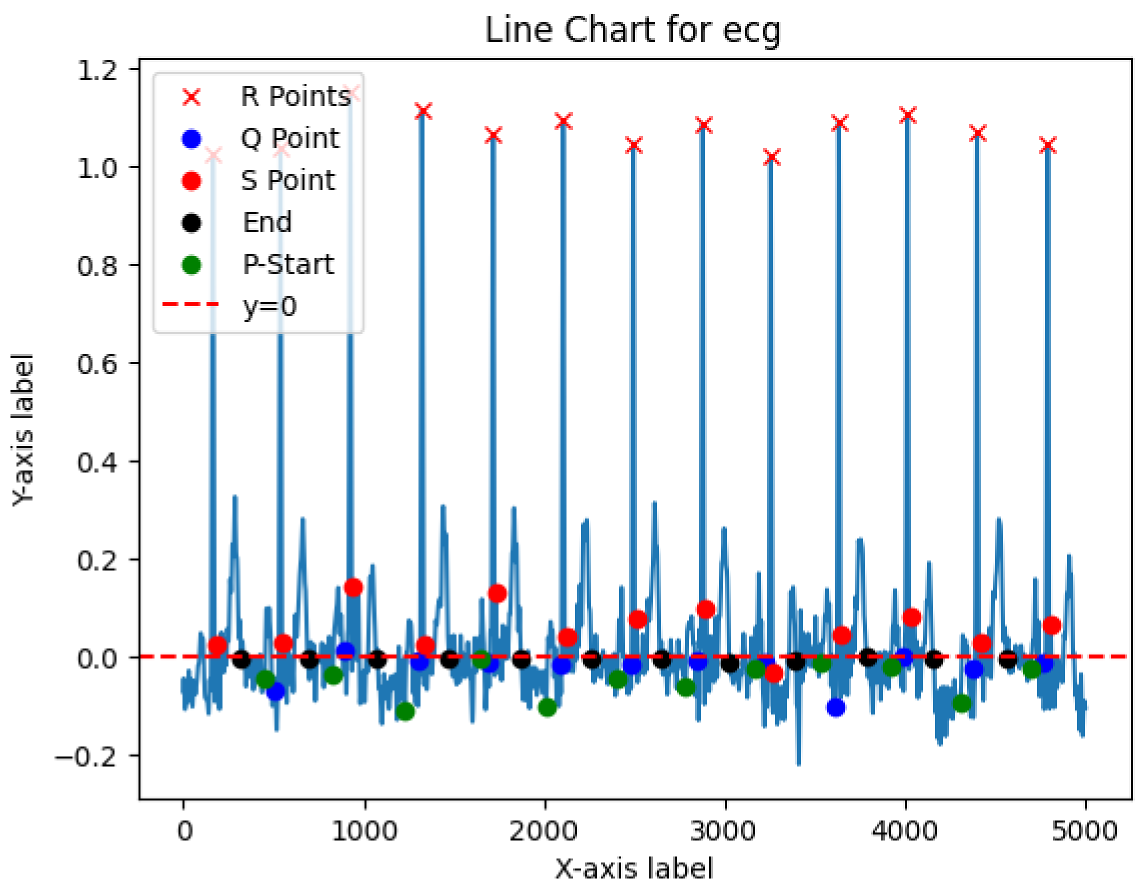
In [ ]:
```
 #Calculation of NEW PR
    # print("Wave-End",wave_end)
    # print("Q-points",newQ_points,len(newQ_points))
newQ_points.pop(0)
```

```
    # print("newQ",newQ_points)
P_points = []

for i in range(len(wave_end)):
    point = find_max_peak_in_range(df, column_name, wave_end[i], newQ_points[i])
    P_points.append(point)
# print("P-points",P_points)
# print("P-points",P_points)
P_pair = dict(filter(lambda x: x is not None, P_points))
# print("P-Pair:",P_pair)
values_list = list(P_pair.values())
    # print("P-peaks",values_list)

p_start=[]
for p_peak in values_list:
    troughs_before_p = [trough for trough in troughs if trough < p_peak]
        # Find the Q point (minima with negative values just before R)
    start_point = max((trough for trough in troughs_before_p if df[column_name][
    p_start.append(start_point)
    # print(p_start)
    # P_start and new Q list should be equal in length
P_dict = {i + 2: value for i, value in enumerate(p_start)}
# print("P-DICT:",P_dict)
if len(newQ_points)>len(p_start):
    newQ_points.pop(0)
PR_durations = []
# result_dict = {key:Q_dict.get(key, None) - P_dict.get(key, None)  if P_dict.ge
# print("P-Q:",result_dict)
# result_values = [value for value in result_dict.values() if value is not None]
# print("PR:",result_values)
for p_point,q_point in zip(p_start, newQ_points):
    if q_point is not None and p_point is not None:
        duration = q_point - p_point
        PR_durations.append(duration)
PR_avg= math.floor(sum(PR_durations)/len(PR_durations))*2
print("PR:",PR_avg)
```

```
PR: 132
```

13. Plotting all the points.

In [ ]:
```
plt.plot(df[column_name])
plt.plot(selected_peaks, df[column_name][selected_peaks], "x", label="R Points",
plt.plot(newQ_points, df[column_name][newQ_points], "o", label="Q Point", color=
plt.plot(newS_points, df[column_name][newS_points], "o", label="S Point", color=
plt.plot(wave_end, df[column_name][wave_end], "o", label="End", color='black')
plt.plot(p_start, df[column_name][p_start], "o", label="P-Start", color='green')
plt.axhline(y=0, color='r', linestyle='--', label="y=0")
# Adding labels and title
plt.xlabel('X-axis label')
plt.ylabel('Y-axis label')
plt.title('Line Chart for {}'.format(column_name))
# Display the chart
plt.legend()
plt.show()
```



Making Data Set

14. Making a Dictionary of all the data that we calculated.

In [ ]:
```
data = {
        "HR": hbpm,
        "R(II)": R_II,
        "RR": RR_Interval,
        "PR": PR_avg,
        "QRS": QRS_avg,
        "QT": QT_avg,
        "QTC": QTC,
        "QT/QC": QT_QC_Ratio
    }
new_df = pd.DataFrame(columns=['HR','R(II)','RR','PR','QRS','QT','QTC','QT/QTC']
row = list(data.values())
new_df.loc[len(new_df)] = row
print(new_df)
```

```
   HR  R(II)   RR   PR  QRS   QT  QTC  QT/QTC
0  77   1.07  770  132   81  347  394    0.88
```