ANALYSIS OF ECG PLOTS USING MACHINE LEARNING TECHNIQUES

Final Year Undergraduate project

in

Computer Science And Engineering

by

Rupesh Das

Roll No:16CS8056

Kunal Dashrath Dekate

Roll No:16CS8101

Under the Guidance of

Prof. Suchismita Roy

National Institute Of Technology, Durgapur

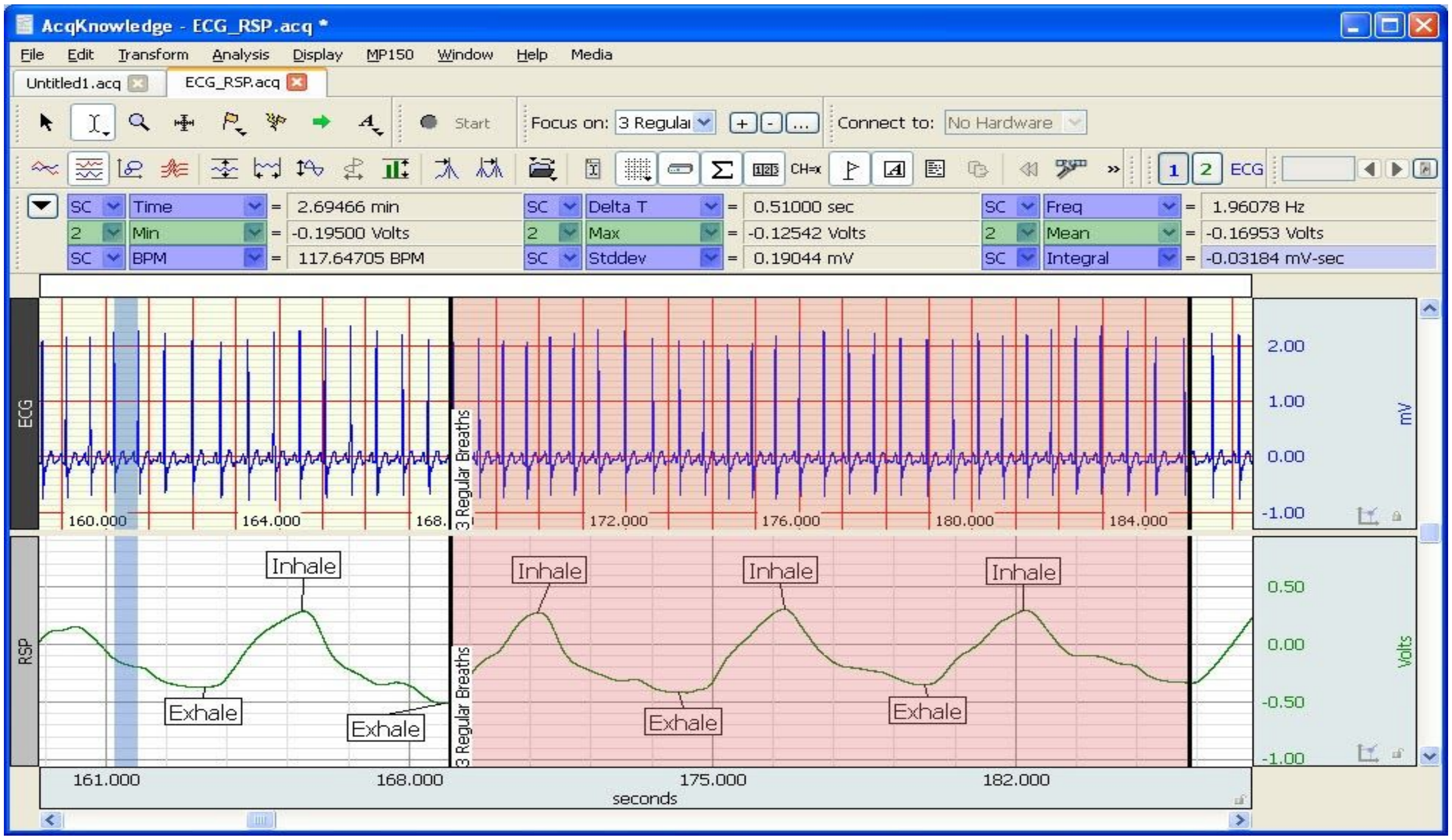
Department of Computer Science And Engineering

INDEX

Introduction

1.What is an ACQ file?

An ACQ file is a data graph file created by AcqKnowledge, a program used to simulate physiological data recorded from different systems and transducers. It contains physiological data recorded from an organism that is displayed in waveforms in AcqKnowledge. ACQ files also store information about how the data was collected, including the rate and duration of the collection. AcqKnowledge is commonly used with BIOPAC hardware, which specializes in collecting data such as EMG, pulse, respiration, EEG, electrogastrogram, temperature, and eye movement of a person.



SC	Time	=	2.69466 min	SC	Delta T	=	0.51000 sec	SC	Freq	=	1.96078 Hz
2	Min	=	-0.19500 Volts	2	Max	=	-0.12542 Volts	2	Mean	=	-0.16953 Volts
SC	BPM	=	117.64705 BPM	SC	Stddev	=	0.19044 mV	SC	Integral	=	-0.03184 mV-sec

Fig. 1 - Represents the data stored in an .acq file

1.1.File layout :

There are two main styles of AcqKnowledge file: compressed and uncompressed. Many of the headers are the same in the two styles; however, the overall file layout is quite different. Whether or not the file is compressed is stored in a byte in the graph header.

Uncompressed

Layout:

```
Graph Header
Channel Header 1
...
Channel Header N
Foreign Data Header
Channel Data Type Header 1
...
Channel Data Type Header N
Interleaved Channel Data
Marker Header
Marker Item 1
...
Marker Item M
Journal Header
Journal Contents
```

Reading uncompressed data

In uncompressed files, the data is interleaved — if you have two channels of data sampled at the same rate, the data will be ordered like 0 1 0 1 0 1 .... Each channel, however, can be sampled as a fraction of the data file's main sampling rate; this fractional rate is the frequency divider for the channel. In AcqKnowledge, this is limited to powers of two. For a data file where channel 0 has a frequency divider of 1 and channel 1 has a frequency divider of 4, the data stream will look like 0 1 0 0 0 0 1 0 0 0 0 1 .... The data can be grouped into a repeating pattern as so: 01000 01000 01000...

Compressed

In compressed files, the data is segregated by channel, and the data channels come after the marker and journal data. Data is compressed with zlib compression, and is little-endian regardless of the endianness of the rest of the file.

Layout:

Graph Header
Channel Header 1
...
Channel Header N
Foreign Data Header
Channel Data Type Header 1
...
Channel Data Type Header N
Marker Header
Marker Item 1
...
Marker Item M
Journal Header
Journal Contents
Channel Compression Header 1
...
Channel Compression Header N

In general, reading the data from compressed files is much easier than uncompressed files. Seek to the end of a compression header, read compressed_data_len bytes, and decompress it into a numpy array.

Markers

Markers can contain a channel number to indicate that it's marking a particular channel of data (or -1 to indicate it's a global marker). This number does not correspond to its index in the interleaved or compressed data, but rather to the order_num field in the channel header, which controls channel display order in the AcqKnowledge UI.

Journal data

Stored after the event markers, the journal is fundamentally just a big block of text. In more recent versions (4.2 +) of AcqKnowledge, it's stored as HTML instead of plain text.

Header index :

Graph Header The settings that apply to an entire file show up here. Both "functional" (eg, msec per sample, total number of channels) and "display" data is stored in this section.

Channel Headers There will be one of these per channel (either raw data or computed) in your data file. Similar to the Graph Header, you'll find both functionally-important and display-only things here.

Channel Datatype Headers One per channel. Defines the datatype of the channel (int or float) and the size (in bytes) of each sample.

Channel Compression Headers One per channel, in compressed files only.

Marker Header Tells us how many event markers we should expect in the file.

Marker Item Headers One of these per event marker in the file.

Journal Header The header telling us about the journal data.

1.2 What is ECG?

Electrocardiography is the process of producing an electrocardiogram (ECG or EKG[]). It is a graph of voltage versus time of the electrical activity of the heart using electrodes placed on the skin. These electrodes detect the small electrical changes that are a consequence of cardiac muscle depolarization followed by repolarization during each cardiac cycle (heartbeat). Changes in the normal ECG pattern occur in numerous cardiac abnormalities, including cardiac rhythm disturbances (such as atrial fibrillation and ventricular tachycardia), inadequate coronary artery blood flow (such as myocardial ischemia and myocardial infarction), and electrolyte disturbances (such as hypokalemia and hyperkalemia).

1.2.1 How does ECGwork?

In order to understand ECG, it's helpful to have a basic understanding of heart activity. The main function of the heart is to pump blood through two circuits:1.Pulmonary circuit: through the lungs to oxygenate the blood and remove carbon dioxide; and2.Systemic circuit: to deliver oxygen and nutrients to tissues and remove carbon dioxide.Because the heart moves blood through two separate circuits, it is sometimes described as a dual pump.In order to beat, the heart needs three types of cells:

1. Rhythm generators, which produce an electrical signal (SA node or normal pacemaker)
2. Conductors to spread the pacemaker signal
3. Contractile cells(myocardium) to mechanically pump blood

1.2.2 Components of the ECG:

The electrical events of the heart (ECG) are usually recorded as a pattern of a baseline (isoelectric line,) broken by a Pwave, a QRScomplex, and a Twave. In addition to the wave components of the ECG, there are intervals and segments (Fig. 2).

1. The isoelectric line is a point of departure of the electrical activity of depolarization and repolarization of the cardiac cycles and indicates periods when the ECG electrodes did not detect electrical activity.

2. An interval is a time measurement that includes waves and/or complexes.

3. A segment is a time measurement that does not include waves and/or complexes.

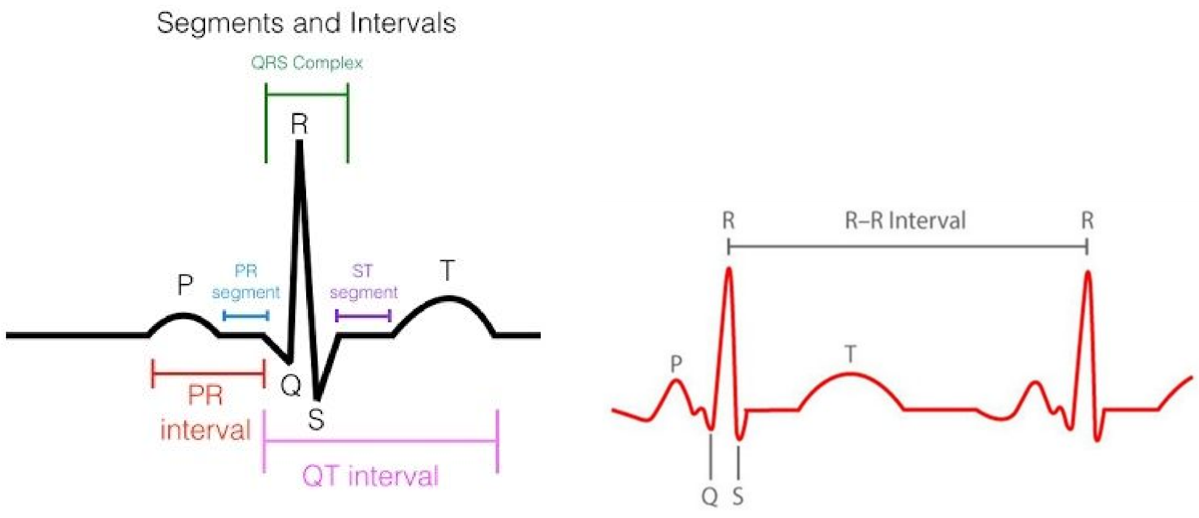


Fig 2.Components of ECG

Algorithm and Procedure

1. First opening the .acq file as this file extension is a special extension which stores medical data using specific instruments only. Using Biopac Api and python we are able to read .acq files and analyse what was stored in it and also in what format.

2. Plotting the graph of channel's data (Desired dependent data Vs. Time) to analyse channels stored in the file.

3. Plotting graph of ECG value vs time . (Graph of PQRS curve) which later on helps in finding " R " peak.

4. Peak Finding Algorithm

5. **Error encountered and solving it** during peak finding algorithm.

6. Linear regression method technique over a **single interval** to find the best curve.

7. Linear regression technique over a **multiple interval** to find the best curve.

Experimental Steps

1.Reading data from xyz.acq file

For the reading of ACQ file we are using bioread library in pythonX. If we want to process the data as numPy array we can use the following api methods as follows

1. Import the necessary libraries such as bioread.

```
import bioread
```

2. Loading the .acq file in a variable ie

```
data=bioread.read_file(“xyz.acq”)
```

3. Finding total numbers of channels

```
n=len(data.channels)
```

4. Finding samples per second in a particular channel

```
data.channels[x].samples_per_second
```

where x=0,1,...,n-1

5. Finding the total number of data in a particular channel

```
len(data.channels[x].data)
```

6. Finding the name of the channel

```
data.channels[x].name
```

7. Description of the channel

```
data.named_channels['channel name’]
```

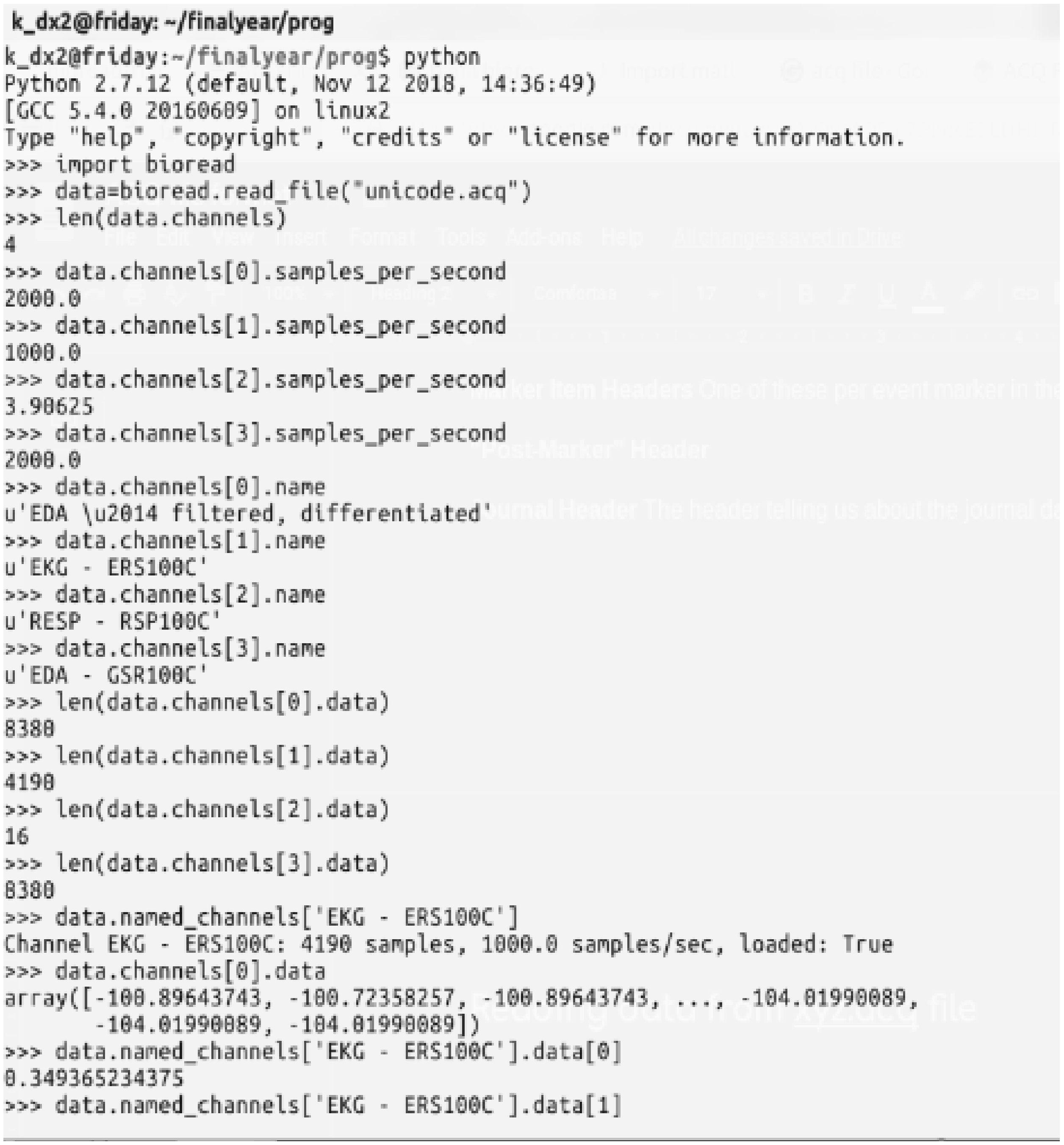
8. The data present at the nth index of the particular channel

   data.named_channels['channel name'].data[n]

9. To display the data of a particular channel in form of array : data.channels[0].data

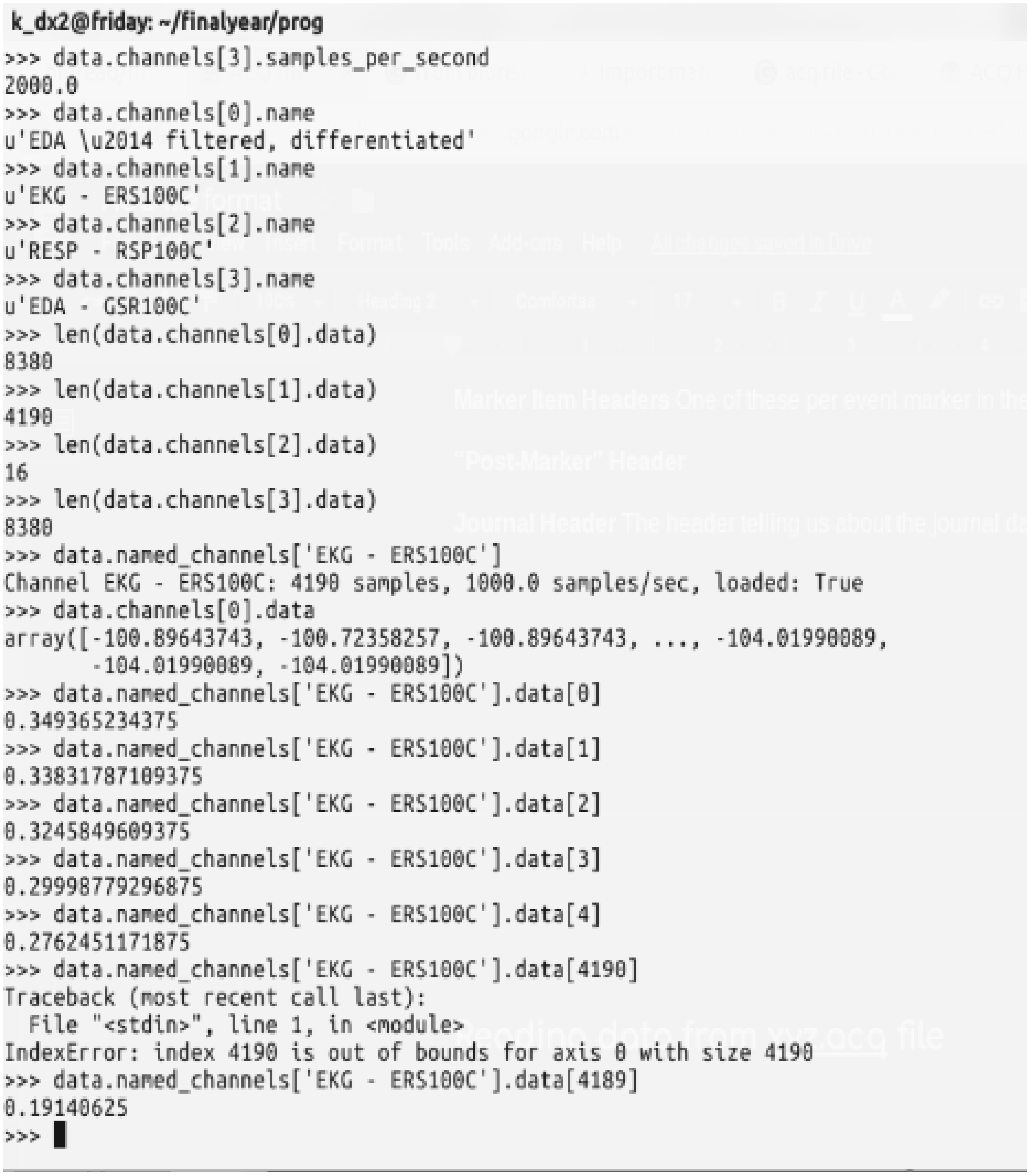
Python Code :

```python
import bioread
import matplotlib.pyplot as plt


data=bioread.read_file("test.acq")

n=len(data.channels)

for i in range (n):
    print('The name of the Channel',data.channels[i].name)
    a=data.channels[i].name
    print('Samples per second',data.channels[i].samples_per_second)
    print('Total number of data',len(data.channels[i].data))
    print('Data',data.channels[i].data)
    plt.plot(data.channels[i].data)
    plt.title(a, fontdict=None, loc='center')
    plt.show()
```



```
k_dx2@friday: ~/finalyear/prog
k_dx2@friday:~/finalyear/prog$ python
Python 2.7.12 (default, Nov 12 2018, 14:36:49)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import bioread
>>> data=bioread.read_file("unicode.acq")
>>> len(data.channels)
4
>>> data.channels[0].samples_per_second
2000.0
>>> data.channels[1].samples_per_second
1000.0
>>> data.channels[2].samples_per_second
3.90625
>>> data.channels[3].samples_per_second
2000.0
>>> data.channels[0].name
u'EDA \u2014 filtered, differentiated'
>>> data.channels[1].name
u'EKG - ERS100C'
>>> data.channels[2].name
u'RESP - RSP100C'
>>> data.channels[3].name
u'EDA - GSR100C'
>>> len(data.channels[0].data)
8380
>>> len(data.channels[1].data)
4190
>>> len(data.channels[2].data)
16
>>> len(data.channels[3].data)
8380
>>> data.named_channels['EKG - ERS100C']
Channel EKG - ERS100C: 4190 samples, 1000.0 samples/sec, loaded: True
>>> data.channels[0].data
array([-100.89643743, -100.72358257, -100.89643743, ..., -104.01990089,
       -104.01990089, -104.01990089])
>>> data.named_channels['EKG - ERS100C'].data[0]
0.349365234375
>>> data.named_channels['EKG - ERS100C'].data[1]
```

(Running code Line by line using Terminal)



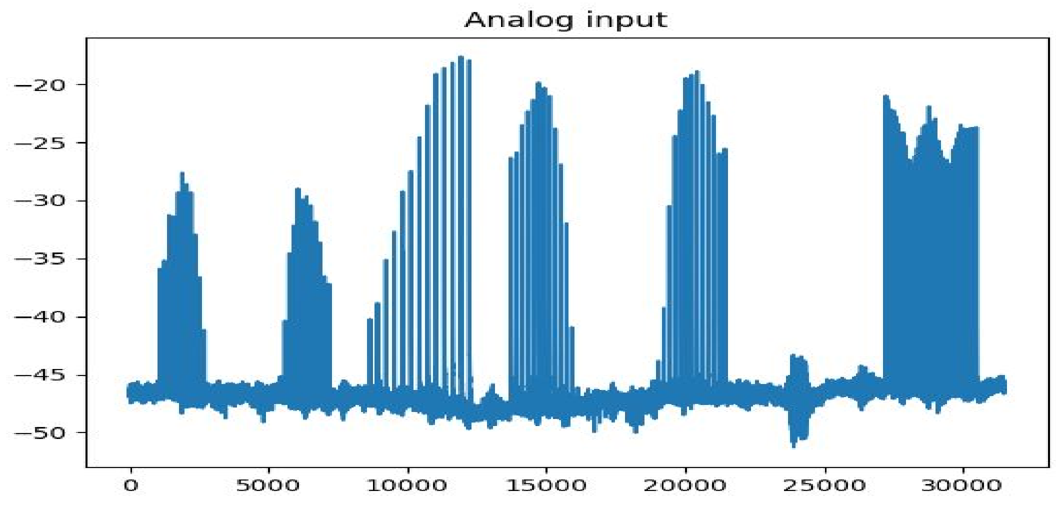
```
k_dx2@friday: ~/finalyear/prog
>>> data.channels[3].samples_per_second
2000.0
>>> data.channels[0].name
u'EDA \u2014 filtered, differentiated'
>>> data.channels[1].name
u'EKG - ERS100C'
>>> data.channels[2].name
u'RESP - RSP100C'
>>> data.channels[3].name
u'EDA - GSR100C'
>>> len(data.channels[0].data)
8380
>>> len(data.channels[1].data)
4190
>>> len(data.channels[2].data)
16
>>> len(data.channels[3].data)
8380
>>> data.named_channels['EKG - ERS100C']
Channel EKG - ERS100C: 4190 samples, 1000.0 samples/sec, loaded: True
>>> data.channels[0].data
array([-100.89643743, -100.72358257, -100.89643743, ..., -104.01990089,
       -104.01990089, -104.01990089])
>>> data.named_channels['EKG - ERS100C'].data[0]
0.349365234375
>>> data.named_channels['EKG - ERS100C'].data[1]
0.33831787109375
>>> data.named_channels['EKG - ERS100C'].data[2]
0.3245849609375
>>> data.named_channels['EKG - ERS100C'].data[3]
0.29998779296875
>>> data.named_channels['EKG - ERS100C'].data[4]
0.2762451171875
>>> data.named_channels['EKG - ERS100C'].data[4190]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: index 4190 is out of bounds for axis 0 with size 4190
>>> data.named_channels['EKG - ERS100C'].data[4189]
0.19140625
>>> 
```

(Running code Line by line using Terminal)

2. Plotting the graph of channel’s data (Desired dependent data Vs. Time) :

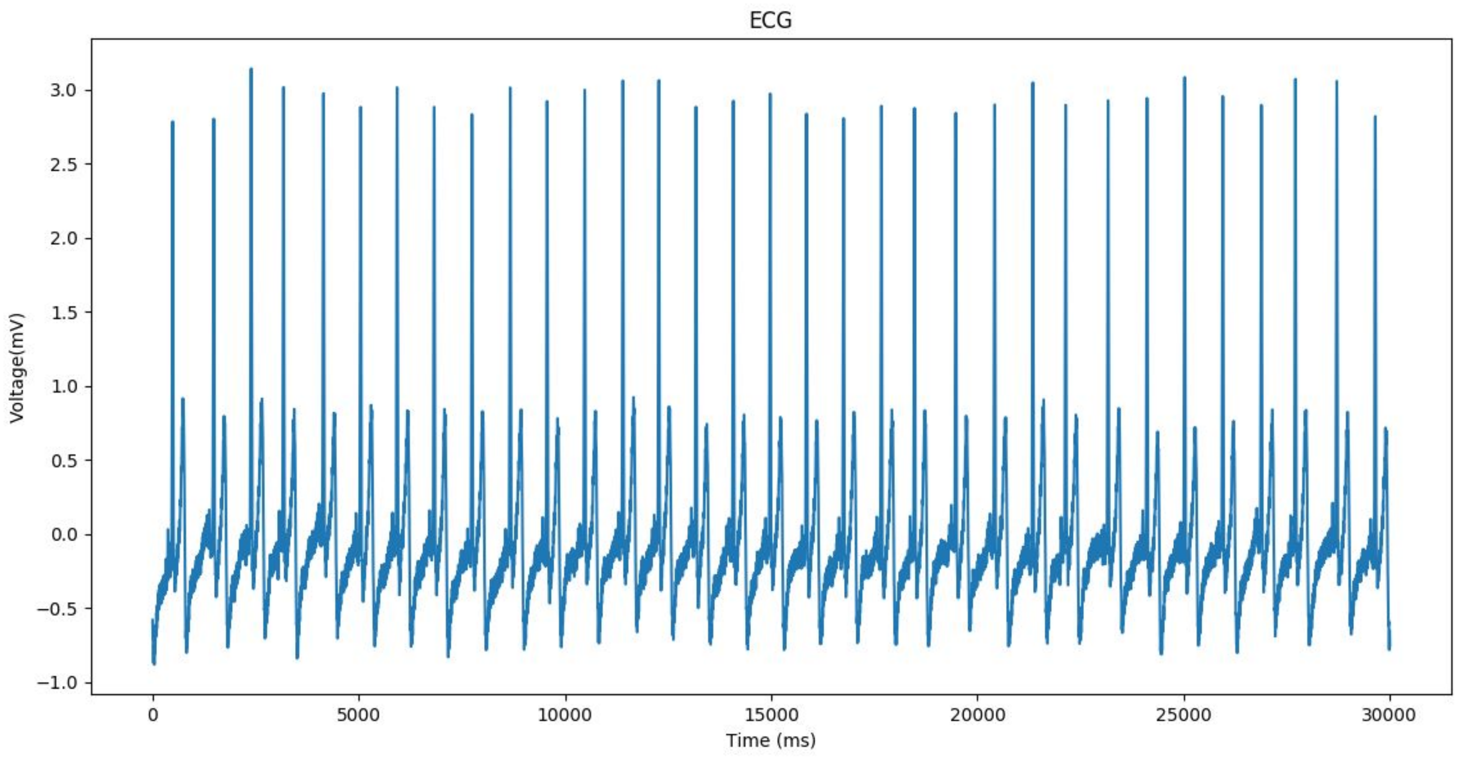
```
plt.plot(data.channels[i].data)

plt.title(channelname, fontdict=None, loc='center')

plt.show()
```

Below are the results obtained from the testing data :



3. ECG :

Analysing the real time data:



Above graph is the graph of **Ecg value vs time .**

4. Peak Finding Algorithm :

Approach: For every element of the array, check whether the current element is a peak (the element has to be greater than its neighbouring elements). We don't have to consider all peaks so by setting the threshold for that particular .acq file as 2.0 we are considering only R peak.

Note that the first and the last element of the array will have a single neighbour.

# Previous Implementation

Python Code

1. Function that returns true if num is greater than both arr[i] and arr[j]

```python
def isPeak(arr, n, num, i, j):
    if arr[i]>2.0:

          if (i >= 0 and arr[i] > num):
              return False

          if (j < n and arr[j] > num):
              return False

          return True
```

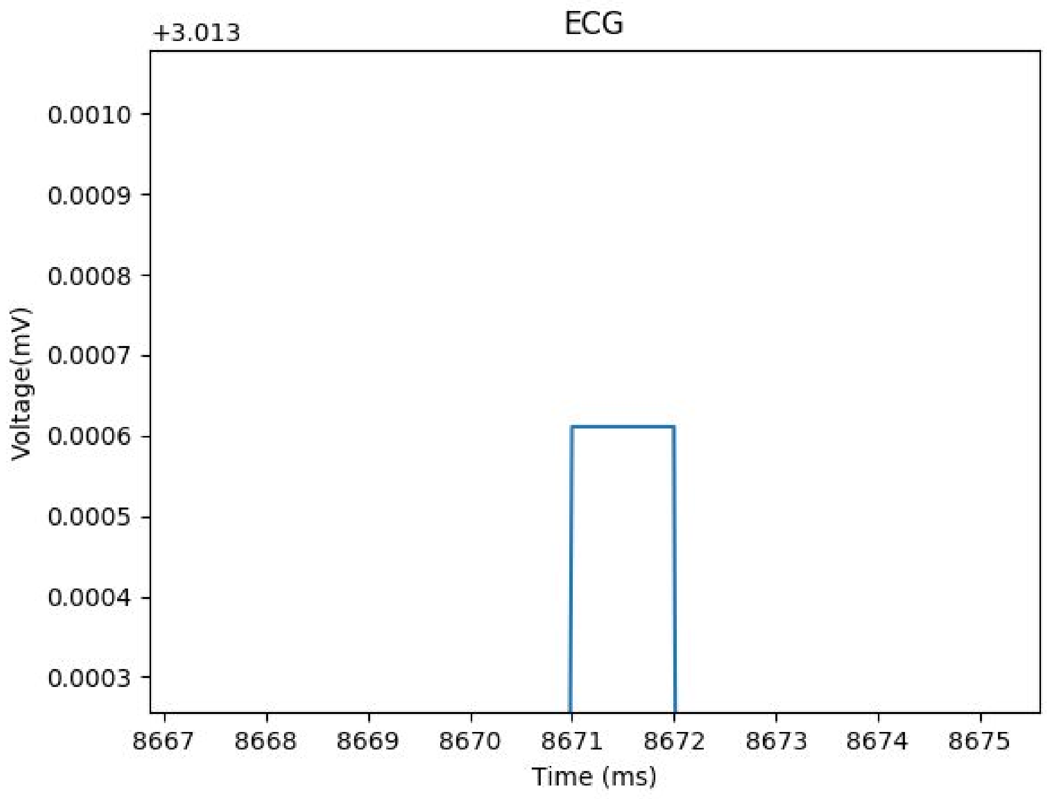
2. Function that appends the Peaks in peak array

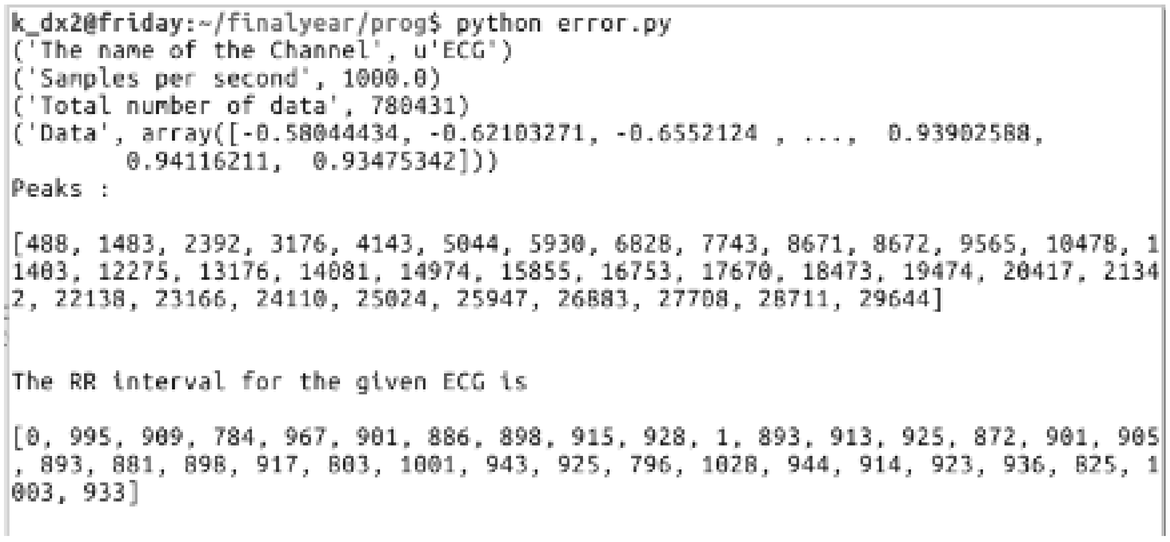
```python
def printPeaks(arr, n):
    print("Peaks : ", end = "")

    for i in range(n):

          if (isPeak(arr, n, arr[i], i - 1, i + 1)):
            peaks.append(i)
```

```
# New Method
r_peaks= detectors.engzee_detector(data.channels[i].data[0:ref_size])


This is for finding Threshold Dynamically
```

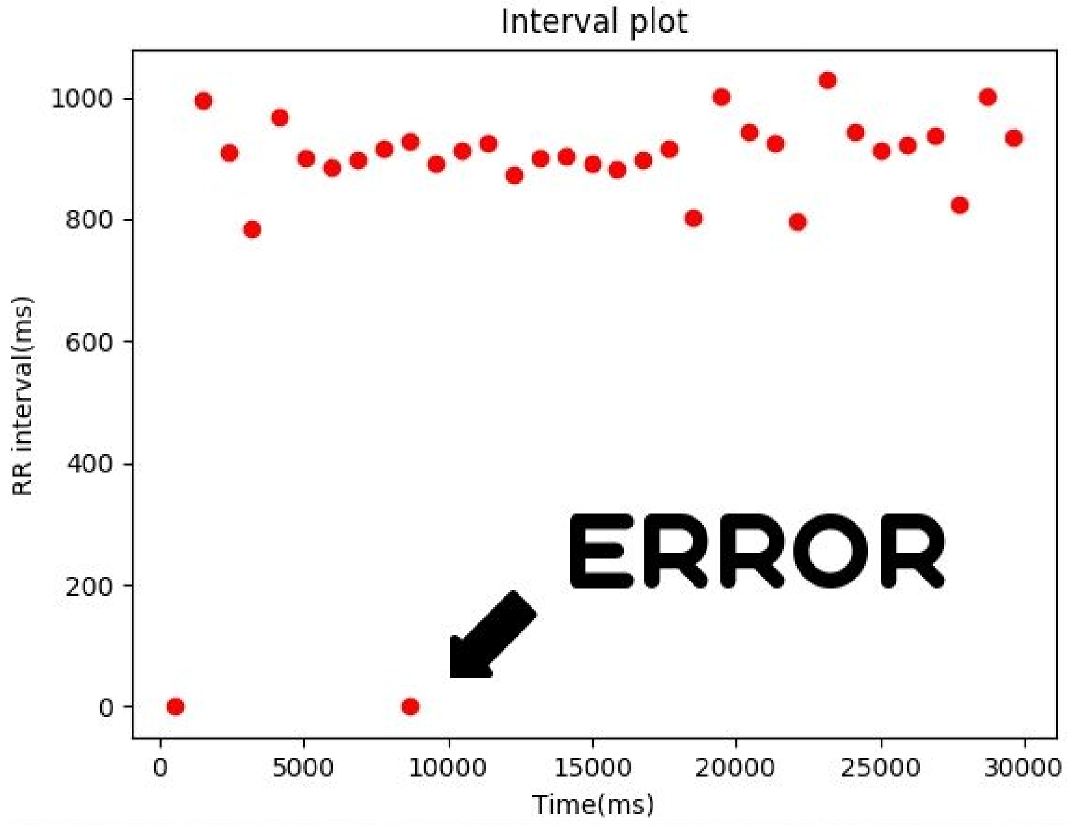
5. Error Encountered during peak Finding Algorithm

In the ecg file the peaks can be obtained as shown below:





```
k_dx2@friday:~/finalyear/prog$ python error.py
('The name of the Channel', u'ECG')
('Samples per second', 1000.0)
('Total number of data', 780431)
('Data', array([-0.58044434, -0.62103271, -0.6552124 , ...,  0.93902588,
        0.94116211,  0.93475342]))
Peaks :

[488, 1483, 2392, 3176, 4143, 5044, 5930, 6828, 7743, 8671, 8672, 9565, 10478, 1
1403, 12275, 13176, 14081, 14974, 15855, 16753, 17670, 18473, 19474, 20417, 2134
2, 22138, 23166, 24110, 25024, 25947, 26883, 27708, 28711, 29644]


The RR interval for the given ECG is

[0, 995, 909, 784, 967, 901, 886, 898, 915, 928, 1, 893, 913, 925, 872, 901, 905
, 893, 881, 898, 917, 803, 1001, 943, 925, 796, 1028, 944, 914, 923, 936, 825, 1
003, 933]
```



According to the algorithm the peaks will be obtained at both 8671 ms and 8672 ms, hence the RR interval will be of 1 ms which is not possible. So we have to consider only one time, which we are considering as the latter one. This might occur many times in the ecg plot. We have to consider a cleaned version of the RR interval array and peaks array (time at which peaks occur).

6. Using Linear regression technique over single interval to find best curve :

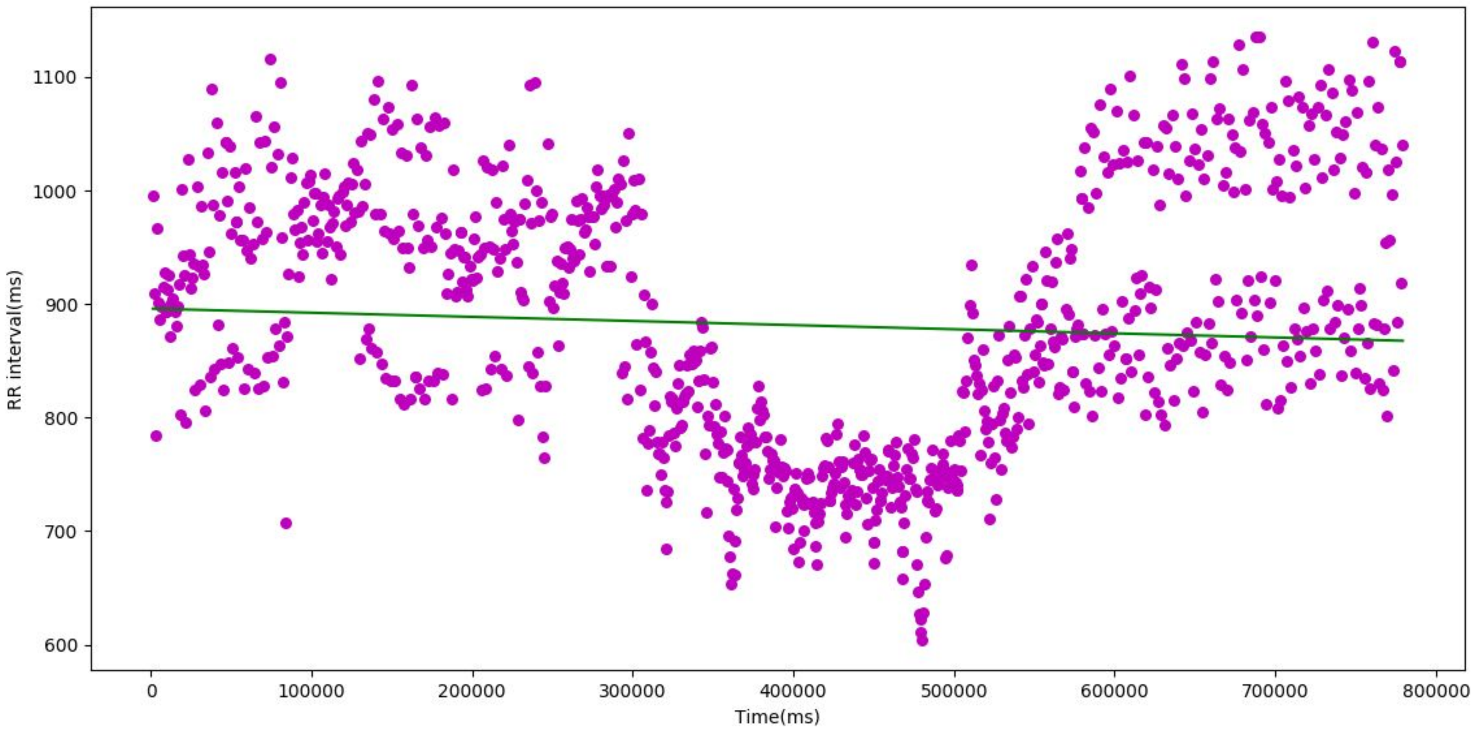
We have used a statistical approach (**Linear regression**) for modelling the relationship between a dependent variable with a given set of independent variables.

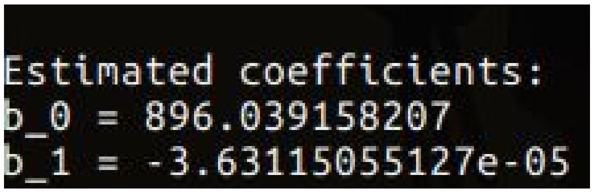
```
# The function def estimate_coef(x , y) -> Calculates
regression coefficient

def estimate_coef(x, y):
    # number of observations/points
    n = np.size(x)

    # mean of x and y vector
    m_x, m_y = np.mean(x), np.mean(y)

    # calculating cross-deviation and deviation about x
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x

    # calculating regression coefficients
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x

    return(b_0, b_1)
```

# The function regression_line(x, y, b) plots the Regression line

```
def plot_regression_line(x, y, b):
    # plotting the actual points as scatter plot
    plt.scatter(x, y, color = "m",marker = "o", s = 30)

    # predicted response vector
    y_pred = b[0] + b[1]*x

    # plotting the regression line
    plt.plot(x, y_pred, color = "g")

    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')
    # function to show plot
    plt.show()


# Driver function
b = estimate_coef(x, y)
     print("Estimated coefficients:\nb_0 = {}  \
          \nb_1 = {}".format(b[0], b[1]))
    plot_regression_line(x, y, b)
```

Above code snippet's Graph obtained :





```
Estimated coefficients:
b_0 = 896.039158207
b_1 = -3.63115055127e-05
```

7. Using Linear regression technique over multiple interval to find best curve :

# The function def estimate_coef(x , y) -> Calculates regression coefficient

```python
def estimate_coef(x, y):
    # number of observations/points
    n = np.size(x)

    # mean of x and y vector
    m_x, m_y = np.mean(x), np.mean(y)

    # calculating cross-deviation and deviation about x
    SS_xy = np.sum(y*x) - n*m_y*m_x
    SS_xx = np.sum(x*x) - n*m_x*m_x

    # calculating regression coefficients
    b_1 = SS_xy / SS_xx
    b_0 = m_y - b_1*m_x
    return(b_0, b_1)
```

# The function regression_line(x, y, b) plots the Regression line

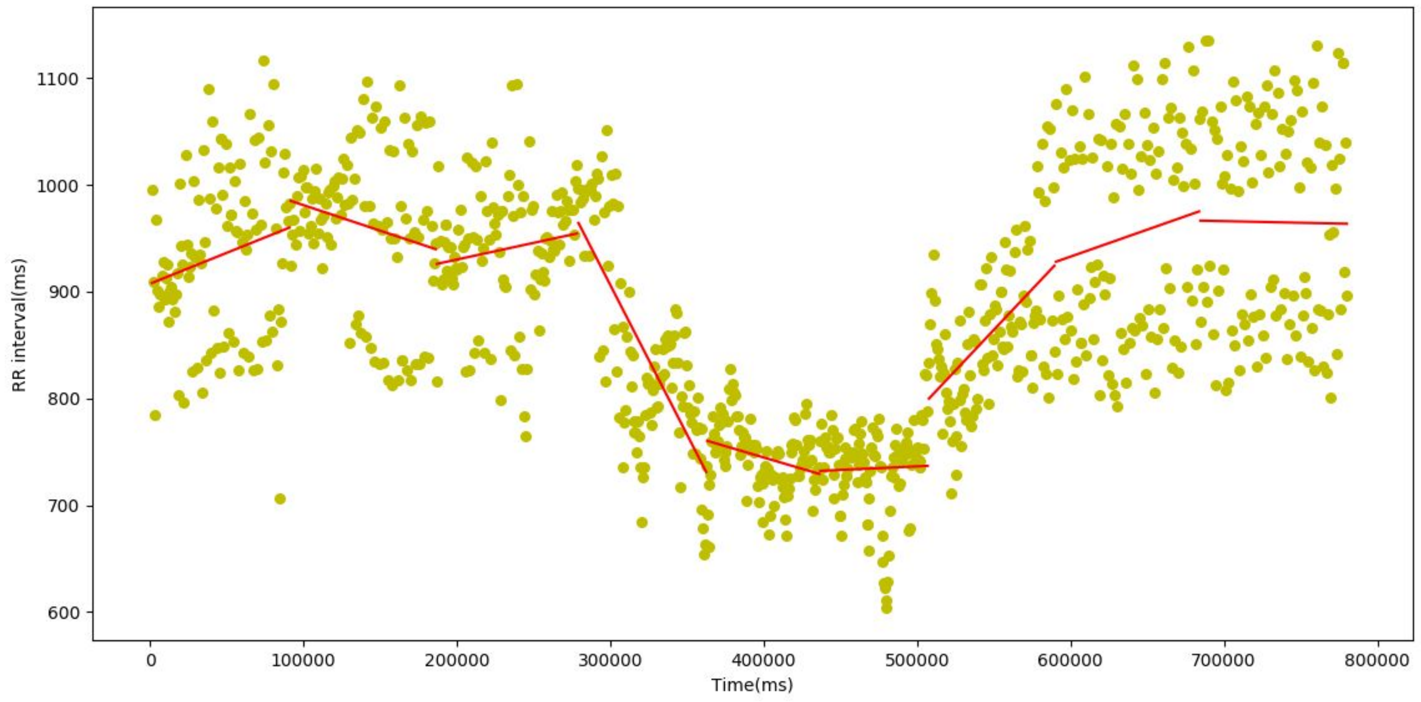
```
def plot_regression_line(x, y, b):
    # plotting the actual points as scatter plot
    plt.scatter(x, y, color = "m",marker = "o", s = 30)

    # predicted response vector
    y_pred = b[0] + b[1]*x

    # plotting the regression line
    plt.plot(x, y_pred, color = "g")

    # putting labels
    plt.xlabel('x')
    plt.ylabel('y')

    # function to show plot
    plt.show()
```

```
# Driver function

     interval=100
     loop=size_x/interval
     i=0
     val=0
     for i in range(loop-1):
            b = estimate_coef(x[val:val+100], y[val:val+100])
            print("Estimated coefficients:\nb_0 = {}\nb_1 =
 }".format(b[0], b[1]))

 plot_regression_line(x[val:val+100], y[val:val+100], b)
           val=val+interval

     b = estimate_coef(x[val:size_x], y[val:size_y])

     print("Estimated coefficients:\nb_0 = {}\nb_1 =
 {}".format(b[0], b[1]))

     plot_regression_line(x[val:size_x], y[val:size_y], b)
```

```
    plt.show()
```

**** The difference between single and multiple interval method is defined by it’s Driver function**

Above code snippet’s Graph obtained :



Python Implementation

```
import bioread
import matplotlib.pyplot as plt
import numpy as np

ref_size=0
peaks=[]
rr=[]

def estimate_coef(x, y):
     n = np.size(x)
     m_x, m_y = np.mean(x), np.mean(y)
     SS_xy = np.sum(y*x) - n*m_y*m_x
     SS_xx = np.sum(x*x) - n*m_x*m_x
     b_1 = SS_xy / SS_xx
     b_0 = m_y - b_1*m_x
     return(b_0, b_1)

def plot_regression_line(x, y, b):
    plt.scatter(x, y, color = "m",marker = "o", s = 30)
    y_pred = b[0] + b[1]*x
    plt.plot(x, y_pred, color = "g")
    plt.xlabel('Time(ms)')
    plt.ylabel('RR interval(ms)')
    plt.show()




def isPeak(arr,n, num, i, j):

    if arr[i]>2.0:

        if (i >= 0 and arr[i] >num):
         return False


        if (j < n and arr[j] >num ):
            return False
        return True


def printPeaks(arr, n):



    for i in range(n):

        if (isPeak(arr, n, arr[i], i - 1, i + 1)):
            peaks.append(i)
```

```
data=bioread.read_file("test1.acq")

n=len(data.channels)

for i in range (n):
     print('The name of the Channel',data.channels[i].name)
     a=data.channels[i].name
     print('Samples per 
second',data.channels[i].samples_per_second)
     print('Total number of data',len(data.channels[i].data))
     print('Data',data.channels[i].data)
     ref_size=len(data.channels[i].data)
     plt.plot(data.channels[i].data[0:ref_size])
     plt.title(a, fontdict=None, loc='center')
     plt.xlabel("Time (ms)")
     plt.ylabel("Voltage(mV)")
     plt.show()


     printPeaks(data.channels[i].data[0:ref_size], ref_size)

     length=len(peaks)
     rr.append(0)

     for i in range(length-1):
          rr.append(peaks[i+1]-peaks[i])

     print("Peaks :\n")
     print(peaks)
     print("\n")
     print("The RR interval for the given ECG is\n")
     print(rr)
     print("\n")
```

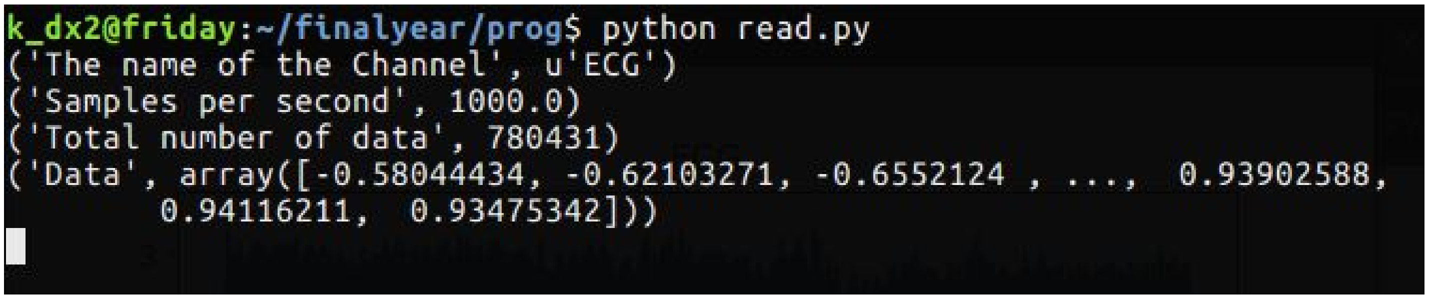
```
#x and y are the cleaned version of  peaks and rr
x=[]
y=[]

for k in range(1,length):
     if rr[k]==1:
          k=k+1

     x.append(peaks[k])
     y.append(rr[k])



print(x)
print("\n")
print(y)
print("\n")
plt.plot(peaks,rr,'ro')
plt.title("Interval plot", fontdict=None, loc='center')
plt.xlabel("Time(ms)")
plt.ylabel("RR interval(ms)")
plt.show()
new_x=np.array(x)
new_y=np.array(y)

#driver code for single interval .
b = estimate_coef(new_x, new_y)
print("Estimated coefficients:\nb_0 = {}  \
     \nb_1 = {}".format(b[0], b[1]))
   plot_regression_line(new_x, new_y, b)
```

For the Python code of Linear Regression over multiple the main difference lies in the driver code where the variable "intervals" defines the number of intervals to be divided and within the loop it calculates regression coefficients and accordingly plots the regression line

```
#driver code for multiple intervals .

      interval=100
     loop=size_x/interval
     i=0
     val=0
     for i in range(loop-1):
          b = estimate_coef(x[val:val+100], y[val:val+100])
          print("Estimated coefficients:\nb_0 = {}\nb_1 =
{}".format(b[0], b[1]))
          plot_regression_line(x[val:val+100], y[val:val+100], b)
          val=val+interval

     b = estimate_coef(x[val:size_x], y[val:size_y])
     print("Estimated coefficients:\nb_0 = {}\nb_1 =
{}".format(b[0], b[1]))
     plot_regression_line(x[val:size_x], y[val:size_y], b)

     plt.show()
```

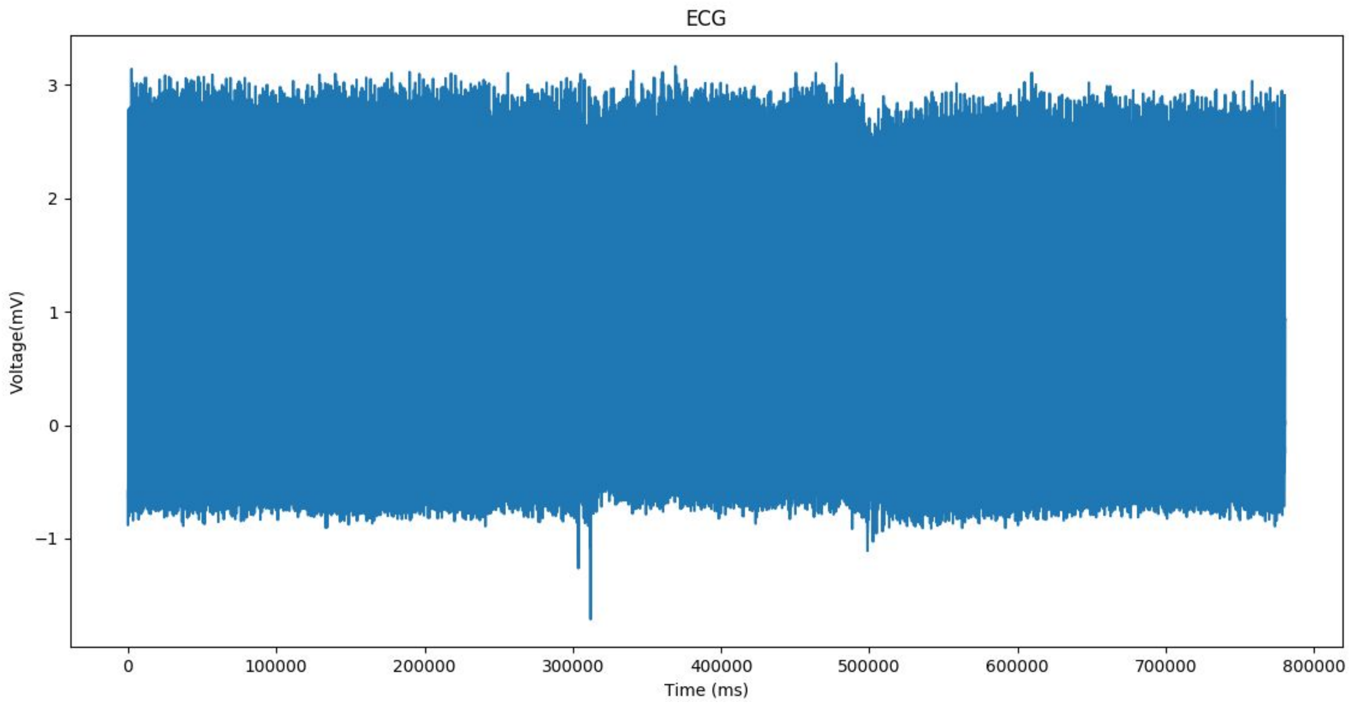
Output and Results

1.) Output for Reading and analysing test1.acq file (Initial Step) :

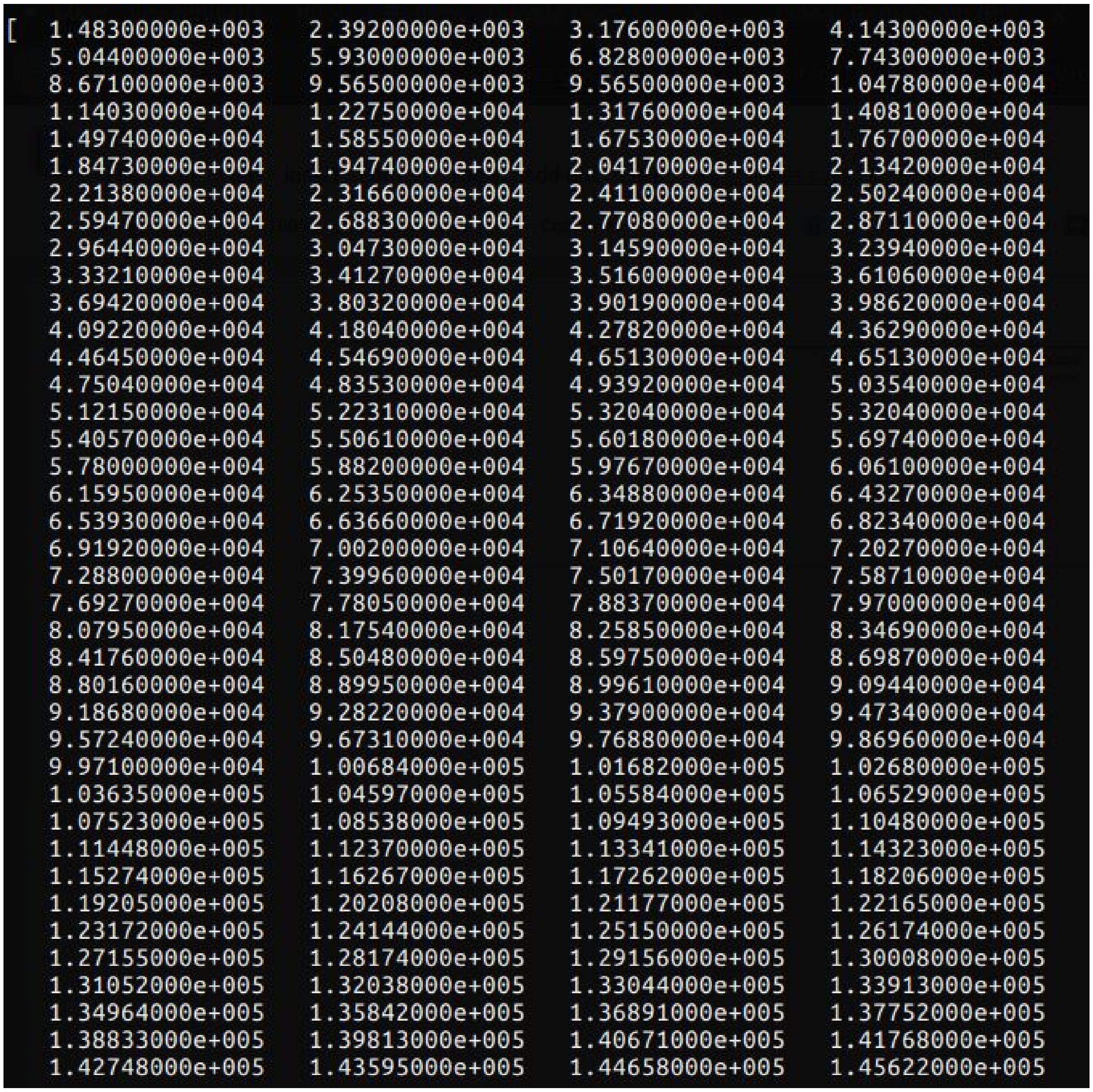


```
k_dx2@friday:~/finalyear/prog$ python read.py
('The name of the Channel', u'ECG')
('Samples per second', 1000.0)
('Total number of data', 780431)
('Data', array([-0.58044434, -0.62103271, -0.6552124 , ...,  0.93902588,
        0.94116211,  0.93475342]))
```

2.) Output for Plot of ECG data :

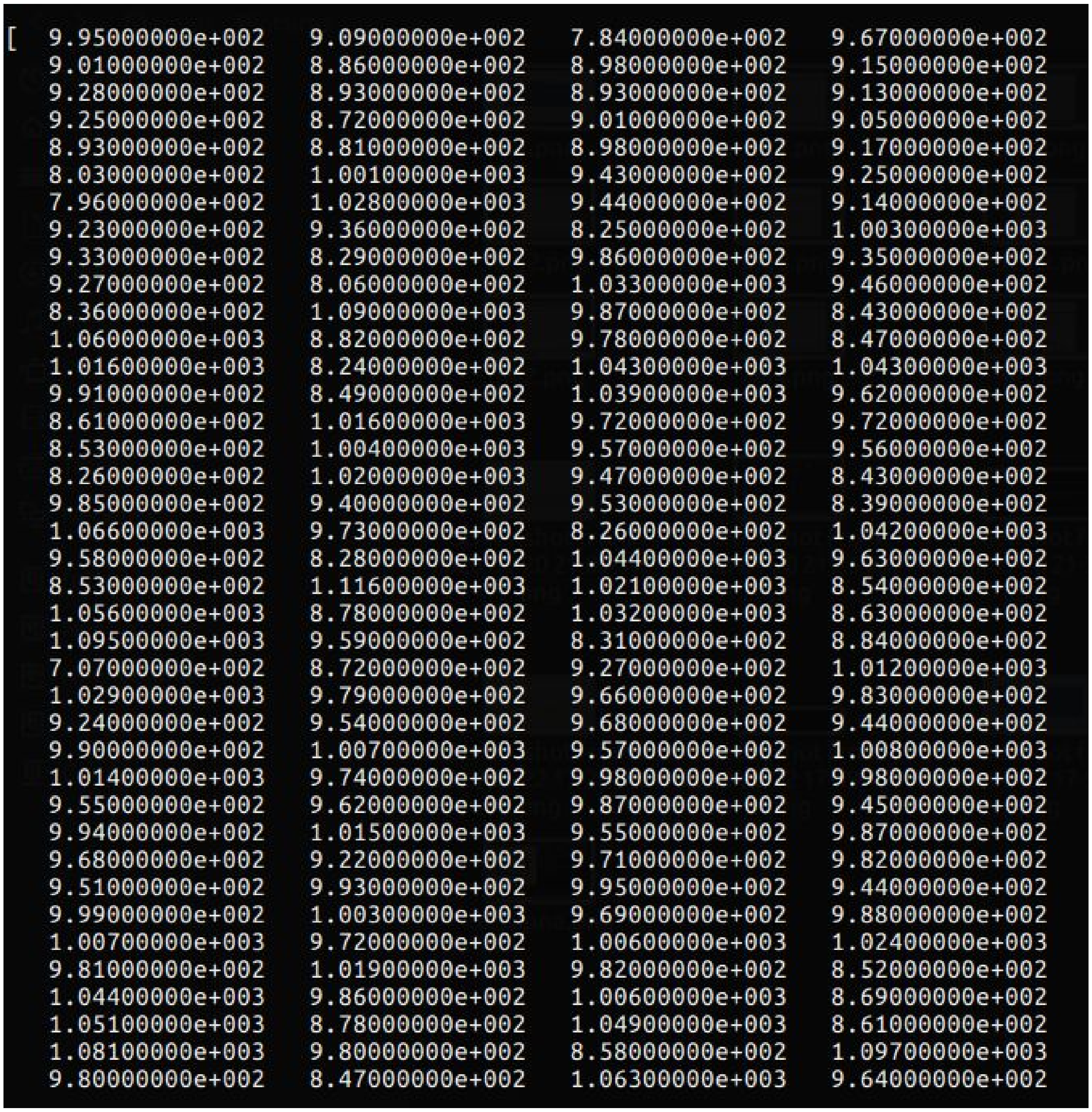


3.) Array of Both R-R interval and it's respective time :



```
[   1.48300000e+003   2.39200000e+003   3.17600000e+003   4.14300000e+003
    5.04400000e+003   5.93000000e+003   6.82800000e+003   7.74300000e+003
    8.67100000e+003   9.56500000e+003   9.56500000e+003   1.04780000e+004
    1.14030000e+004   1.22750000e+004   1.31760000e+004   1.40810000e+004
    1.49740000e+004   1.58550000e+004   1.67530000e+004   1.76700000e+004
    1.84730000e+004   1.94740000e+004   2.04170000e+004   2.13420000e+004
    2.21380000e+004   2.31660000e+004   2.41100000e+004   2.50240000e+004
    2.59470000e+004   2.68830000e+004   2.77080000e+004   2.87110000e+004
    2.96440000e+004   3.04730000e+004   3.14590000e+004   3.23940000e+004
    3.33210000e+004   3.41270000e+004   3.51600000e+004   3.61060000e+004
    3.69420000e+004   3.80320000e+004   3.90190000e+004   3.98620000e+004
    4.09220000e+004   4.18040000e+004   4.27820000e+004   4.36290000e+004
    4.46450000e+004   4.54690000e+004   4.65130000e+004   4.65130000e+004
    4.75040000e+004   4.83530000e+004   4.93920000e+004   5.03540000e+004
    5.12150000e+004   5.22310000e+004   5.32040000e+004   5.32040000e+004
    5.40570000e+004   5.50610000e+004   5.60180000e+004   5.69740000e+004
    5.78000000e+004   5.88200000e+004   5.97670000e+004   6.06100000e+004
    6.15950000e+004   6.25350000e+004   6.34880000e+004   6.43270000e+004
    6.53930000e+004   6.63660000e+004   6.71920000e+004   6.82340000e+004
    6.91920000e+004   7.00200000e+004   7.10640000e+004   7.20270000e+004
    7.28800000e+004   7.39960000e+004   7.50170000e+004   7.58710000e+004
    7.69270000e+004   7.78050000e+004   7.88370000e+004   7.97000000e+004
    8.07950000e+004   8.17540000e+004   8.25850000e+004   8.34690000e+004
    8.41760000e+004   8.50480000e+004   8.59750000e+004   8.69870000e+004
    8.80160000e+004   8.89950000e+004   8.99610000e+004   9.09440000e+004
    9.18680000e+004   9.28220000e+004   9.37900000e+004   9.47340000e+004
    9.57240000e+004   9.67310000e+004   9.76880000e+004   9.86960000e+004
    9.97100000e+004   1.00684000e+005   1.01682000e+005   1.02680000e+005
    1.03635000e+005   1.04597000e+005   1.05584000e+005   1.06529000e+005
    1.07523000e+005   1.08538000e+005   1.09493000e+005   1.10480000e+005
    1.11448000e+005   1.12370000e+005   1.13341000e+005   1.14323000e+005
    1.15274000e+005   1.16267000e+005   1.17262000e+005   1.18206000e+005
    1.19205000e+005   1.20208000e+005   1.21177000e+005   1.22165000e+005
    1.23172000e+005   1.24144000e+005   1.25150000e+005   1.26174000e+005
    1.27155000e+005   1.28174000e+005   1.29156000e+005   1.30008000e+005
    1.31052000e+005   1.32038000e+005   1.33044000e+005   1.33913000e+005
    1.34964000e+005   1.35842000e+005   1.36891000e+005   1.37752000e+005
    1.38833000e+005   1.39813000e+005   1.40671000e+005   1.41768000e+005
    1.42748000e+005   1.43595000e+005   1.44658000e+005   1.45622000e+005
```

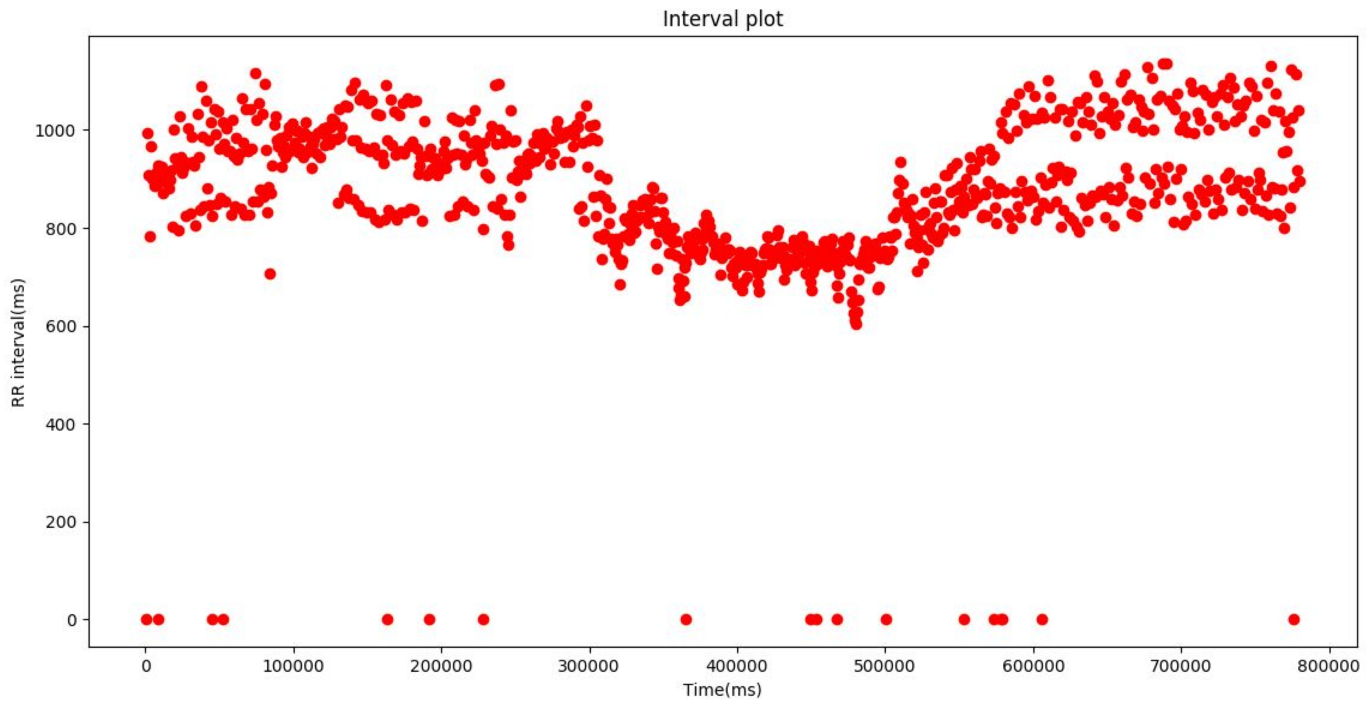
R-R Interval



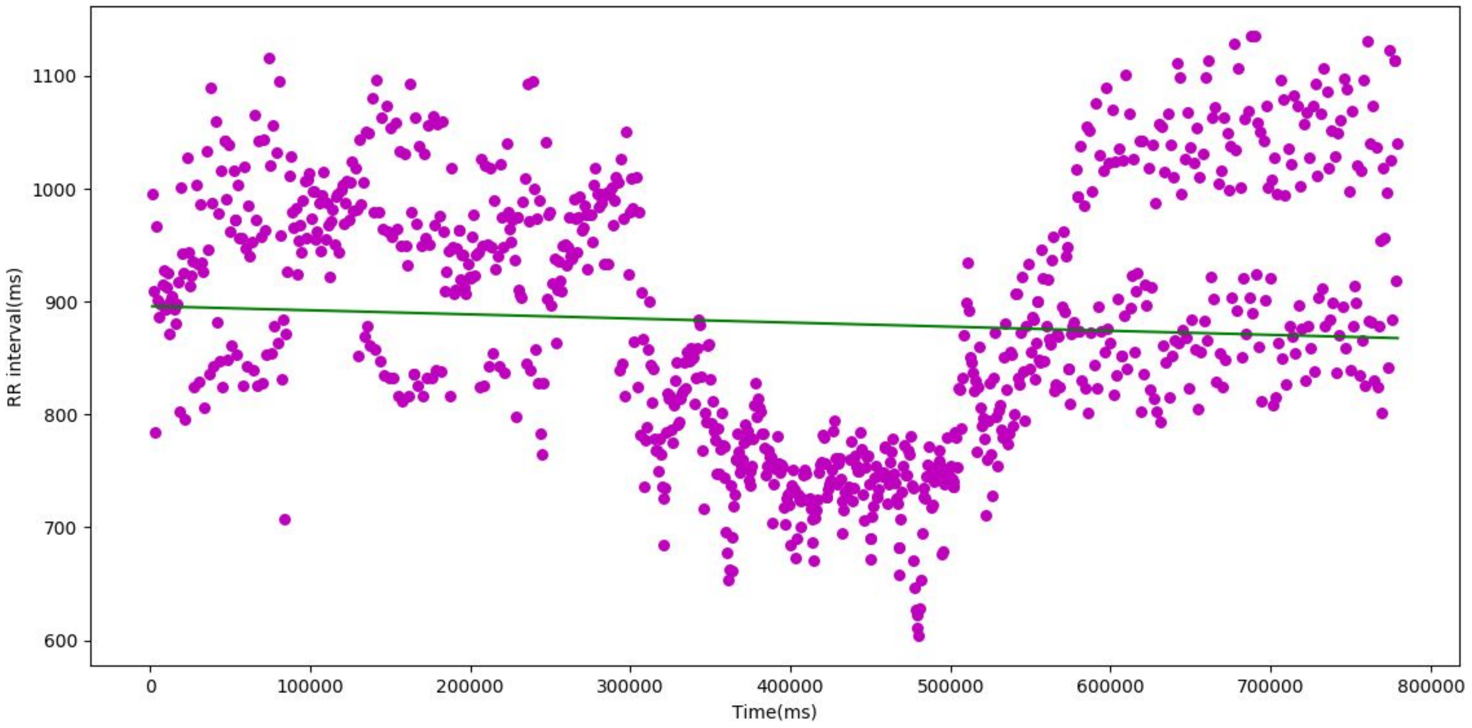
```
[  9.95000000e+002   9.09000000e+002   7.84000000e+002   9.67000000e+002
   9.01000000e+002   8.86000000e+002   8.98000000e+002   9.15000000e+002
   9.28000000e+002   8.93000000e+002   8.93000000e+002   9.13000000e+002
   9.25000000e+002   8.72000000e+002   9.01000000e+002   9.05000000e+002
   8.93000000e+002   8.81000000e+002   8.98000000e+002   9.17000000e+002
   8.03000000e+002   1.00100000e+003   9.43000000e+002   9.25000000e+002
   7.96000000e+002   1.02800000e+003   9.44000000e+002   9.14000000e+002
   9.23000000e+002   9.36000000e+002   8.25000000e+002   1.00300000e+003
   9.33000000e+002   8.29000000e+002   9.86000000e+002   9.35000000e+002
   9.27000000e+002   8.06000000e+002   1.03300000e+003   9.46000000e+002
   8.36000000e+002   1.09000000e+003   9.87000000e+002   8.43000000e+002
   1.06000000e+003   8.82000000e+002   9.78000000e+002   8.47000000e+002
   1.01600000e+003   8.24000000e+002   1.04300000e+003   1.04300000e+003
   9.91000000e+002   8.49000000e+002   1.03900000e+003   9.62000000e+002
   8.61000000e+002   1.01600000e+003   9.72000000e+002   9.72000000e+002
   8.53000000e+002   1.00400000e+003   9.57000000e+002   9.56000000e+002
   8.26000000e+002   1.02000000e+003   9.47000000e+002   8.43000000e+002
   9.85000000e+002   9.40000000e+002   9.53000000e+002   8.39000000e+002
   1.06600000e+003   9.73000000e+002   8.26000000e+002   1.04200000e+003
   9.58000000e+002   8.28000000e+002   1.04400000e+003   9.63000000e+002
   8.53000000e+002   1.11600000e+003   1.02100000e+003   8.54000000e+002
   1.05600000e+003   8.78000000e+002   1.03200000e+003   8.63000000e+002
   1.09500000e+003   9.59000000e+002   8.31000000e+002   8.84000000e+002
   7.07000000e+002   8.72000000e+002   9.27000000e+002   1.01200000e+003
   1.02900000e+003   9.79000000e+002   9.66000000e+002   9.83000000e+002
   9.24000000e+002   9.54000000e+002   9.68000000e+002   9.44000000e+002
   9.90000000e+002   1.00700000e+003   9.57000000e+002   1.00800000e+003
   1.01400000e+003   9.74000000e+002   9.98000000e+002   9.98000000e+002
   9.55000000e+002   9.62000000e+002   9.87000000e+002   9.45000000e+002
   9.94000000e+002   1.01500000e+003   9.55000000e+002   9.87000000e+002
   9.68000000e+002   9.22000000e+002   9.71000000e+002   9.82000000e+002
   9.51000000e+002   9.93000000e+002   9.95000000e+002   9.44000000e+002
   9.99000000e+002   1.00300000e+003   9.69000000e+002   9.88000000e+002
   1.00700000e+003   9.72000000e+002   1.00600000e+003   1.02400000e+003
   9.81000000e+002   1.01900000e+003   9.82000000e+002   8.52000000e+002
   1.04400000e+003   9.86000000e+002   1.00600000e+003   8.69000000e+002
   1.05100000e+003   8.78000000e+002   1.04900000e+003   8.61000000e+002
   1.08100000e+003   9.80000000e+002   8.58000000e+002   1.09700000e+003
   9.80000000e+002   8.47000000e+002   1.06300000e+003   9.64000000e+002
```

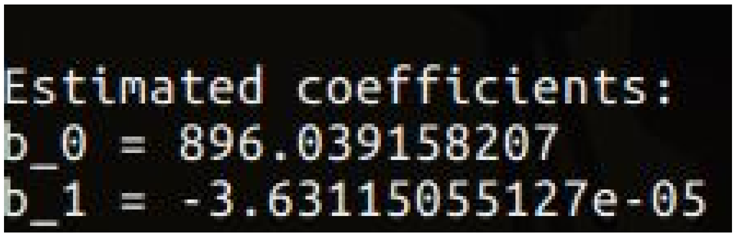
TIME

4.) Output for R-R interval Vs Time (Error included) :



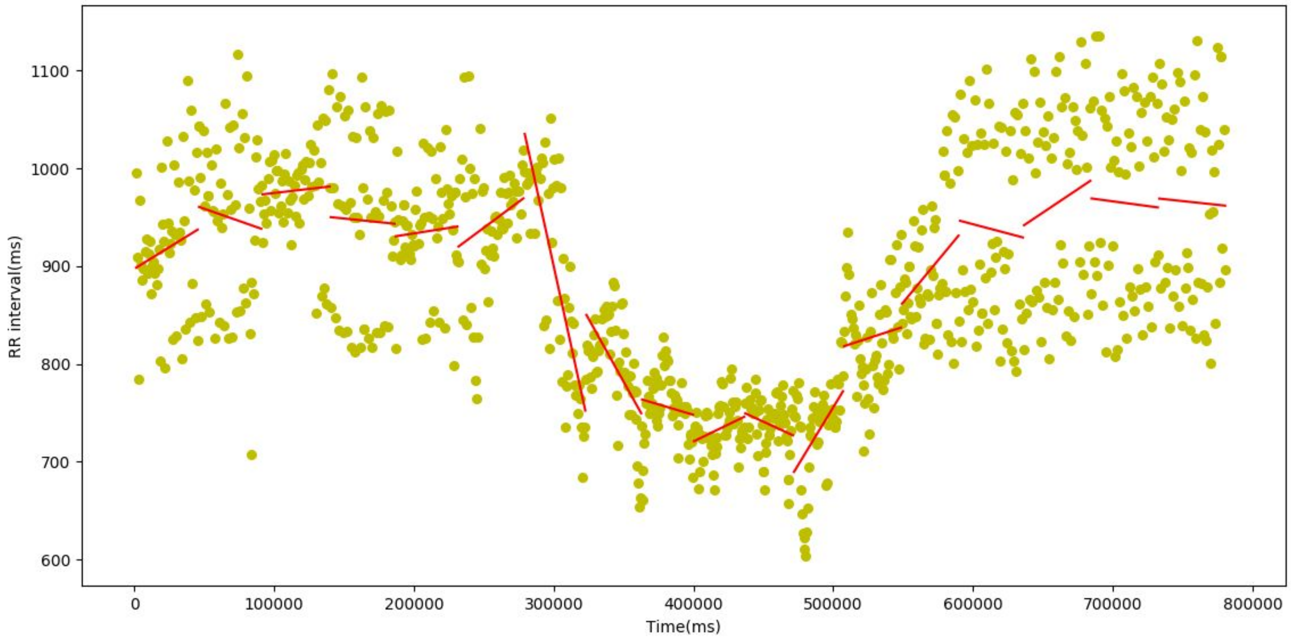
5.) Output for Linear Regression over Single Interval :



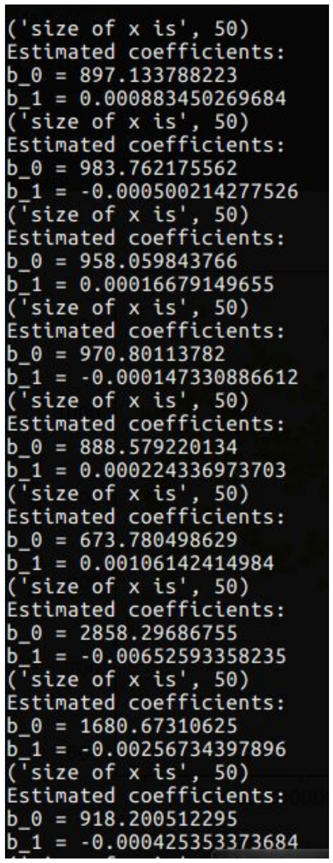


```
Estimated coefficients:
b_0 = 896.039158207
b_1 = -3.63115055127e-05
```

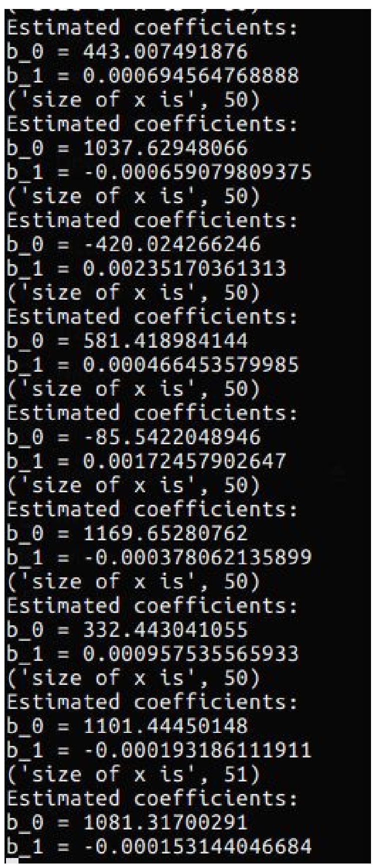
6.) Output for Linear Regression over multiple Interval :



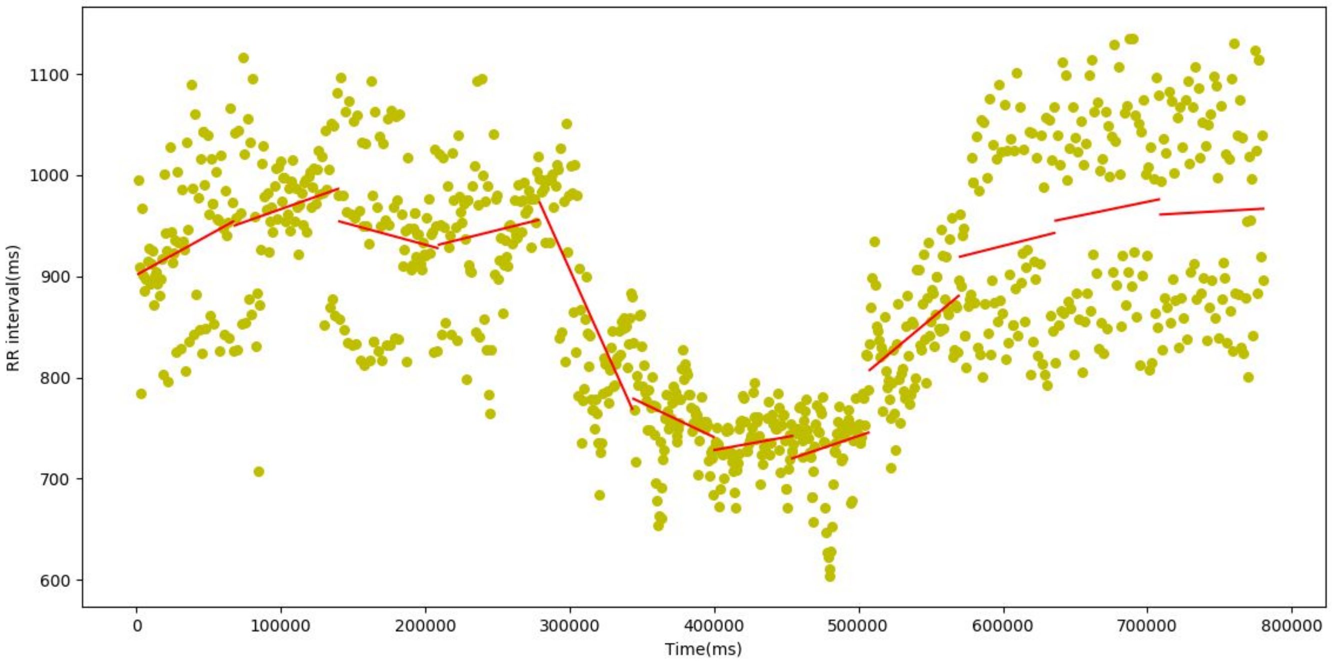
For interval 50



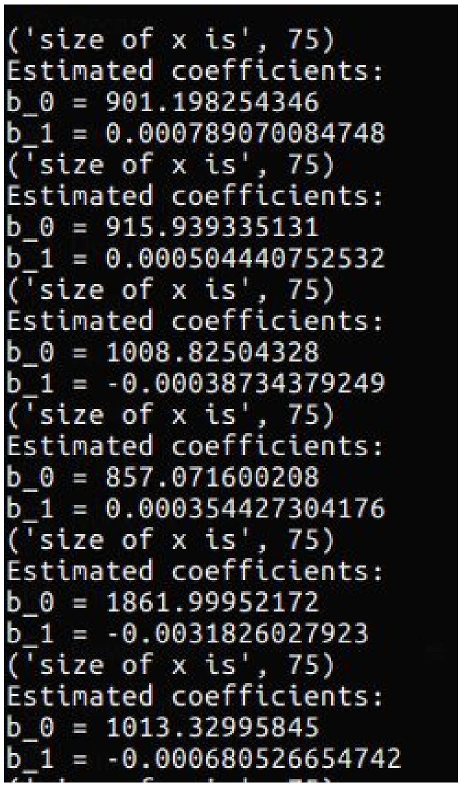
```
('size of x is', 50)
Estimated coefficients:
b_0 = 897.133788223
b_1 = 0.000883450269684
('size of x is', 50)
Estimated coefficients:
b_0 = 983.762175562
b_1 = -0.000500214277526
('size of x is', 50)
Estimated coefficients:
b_0 = 958.059843766
b_1 = 0.00016679149655
('size of x is', 50)
Estimated coefficients:
b_0 = 970.80113782
b_1 = -0.000147330886612
('size of x is', 50)
Estimated coefficients:
b_0 = 888.579220134
b_1 = 0.000224336973703
('size of x is', 50)
Estimated coefficients:
b_0 = 673.780498629
b_1 = 0.00106142414984
('size of x is', 50)
Estimated coefficients:
b_0 = 2858.29686755
b_1 = -0.00652593358235
('size of x is', 50)
Estimated coefficients:
b_0 = 1680.67310625
b_1 = -0.00256734397896
('size of x is', 50)
Estimated coefficients:
b_0 = 918.200512295
b_1 = -0.000425353373684
```



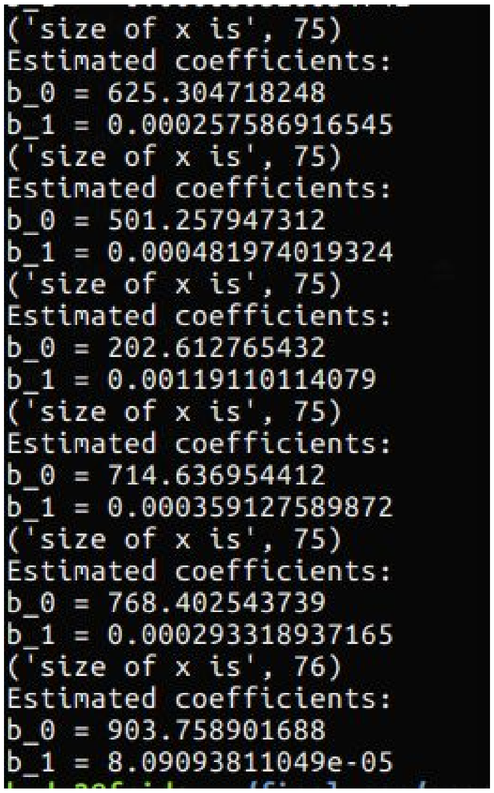
```
Estimated coefficients:
b_0 = 443.007491876
b_1 = 0.000694564768888
('size of x is', 50)
Estimated coefficients:
b_0 = 1037.62948066
b_1 = -0.000659079809375
('size of x is', 50)
Estimated coefficients:
b_0 = -420.024266246
b_1 = 0.00235170361313
('size of x is', 50)
Estimated coefficients:
b_0 = 581.418984144
b_1 = 0.000466453579985
('size of x is', 50)
Estimated coefficients:
b_0 = -85.5422048946
b_1 = 0.00172457902647
('size of x is', 50)
Estimated coefficients:
b_0 = 1169.65280762
b_1 = -0.000378062135899
('size of x is', 50)
Estimated coefficients:
b_0 = 332.443041055
b_1 = 0.000957535565933
('size of x is', 50)
Estimated coefficients:
b_0 = 1101.44450148
b_1 = -0.000193186111911
('size of x is', 51)
Estimated coefficients:
b_0 = 1081.31700291
b_1 = -0.000153144046684
```



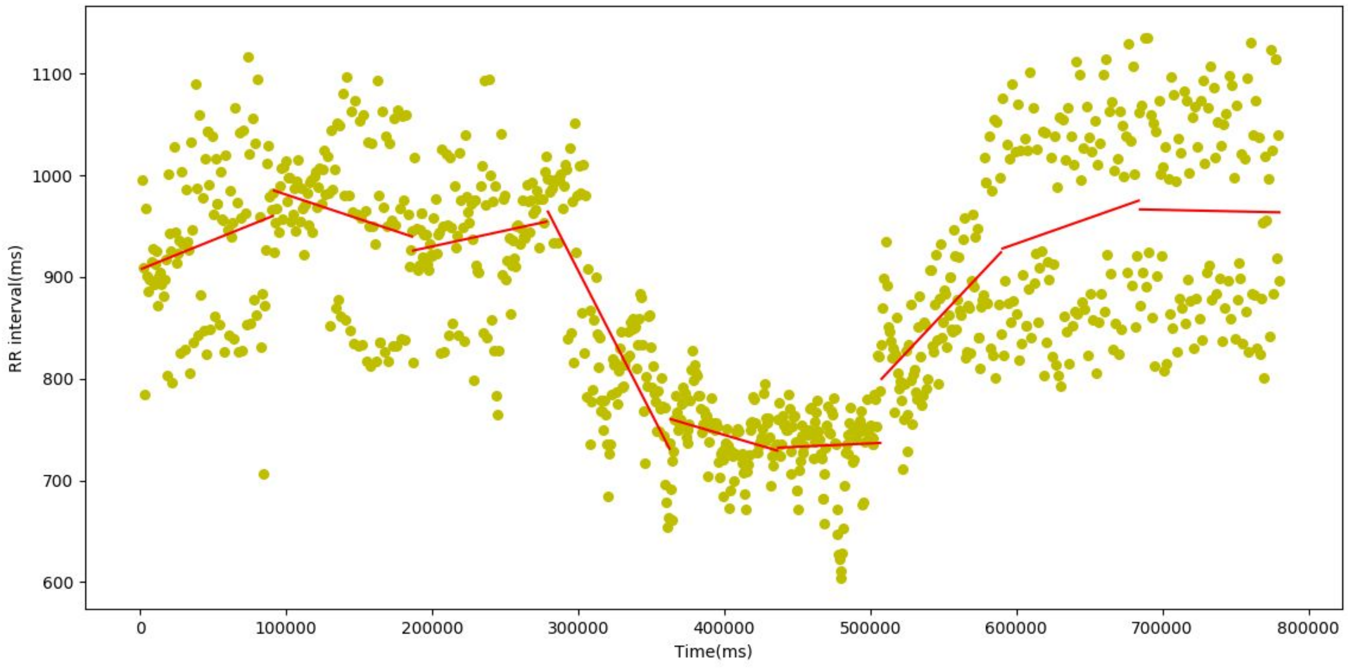
For interval 75



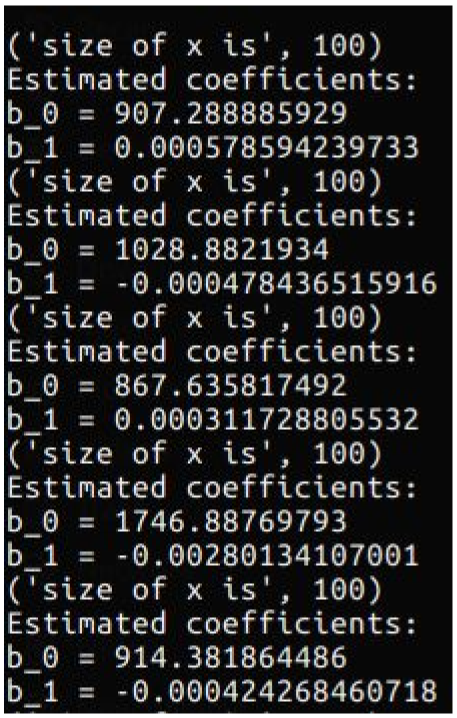
```
('size of x is', 75)
Estimated coefficients:
b_0 = 901.198254346
b_1 = 0.000789070084748
('size of x is', 75)
Estimated coefficients:
b_0 = 915.939335131
b_1 = 0.000504440752532
('size of x is', 75)
Estimated coefficients:
b_0 = 1008.82504328
b_1 = -0.00038734379249
('size of x is', 75)
Estimated coefficients:
b_0 = 857.071600208
b_1 = 0.000354427304176
('size of x is', 75)
Estimated coefficients:
b_0 = 1861.99952172
b_1 = -0.0031826027923
('size of x is', 75)
Estimated coefficients:
b_0 = 1013.32995845
b_1 = -0.000680526654742
```



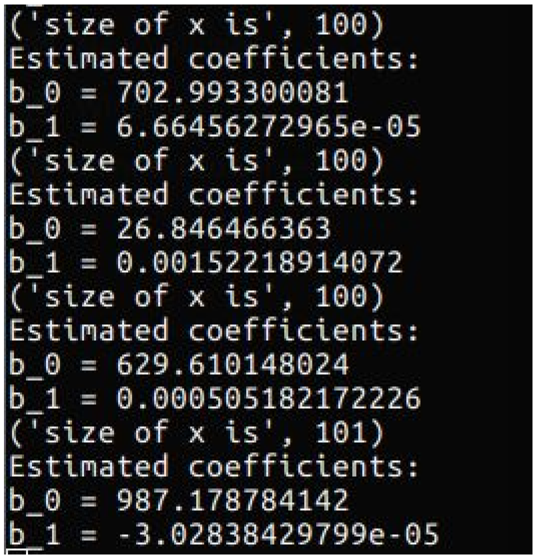
```
('size of x is', 75)
Estimated coefficients:
b_0 = 625.304718248
b_1 = 0.000257586916545
('size of x is', 75)
Estimated coefficients:
b_0 = 501.257947312
b_1 = 0.000481974019324
('size of x is', 75)
Estimated coefficients:
b_0 = 202.612765432
b_1 = 0.00119110114079
('size of x is', 75)
Estimated coefficients:
b_0 = 714.636954412
b_1 = 0.000359127589872
('size of x is', 75)
Estimated coefficients:
b_0 = 768.402543739
b_1 = 0.000293318937165
('size of x is', 76)
Estimated coefficients:
b_0 = 903.758901688
b_1 = 8.09093811049e-05
```



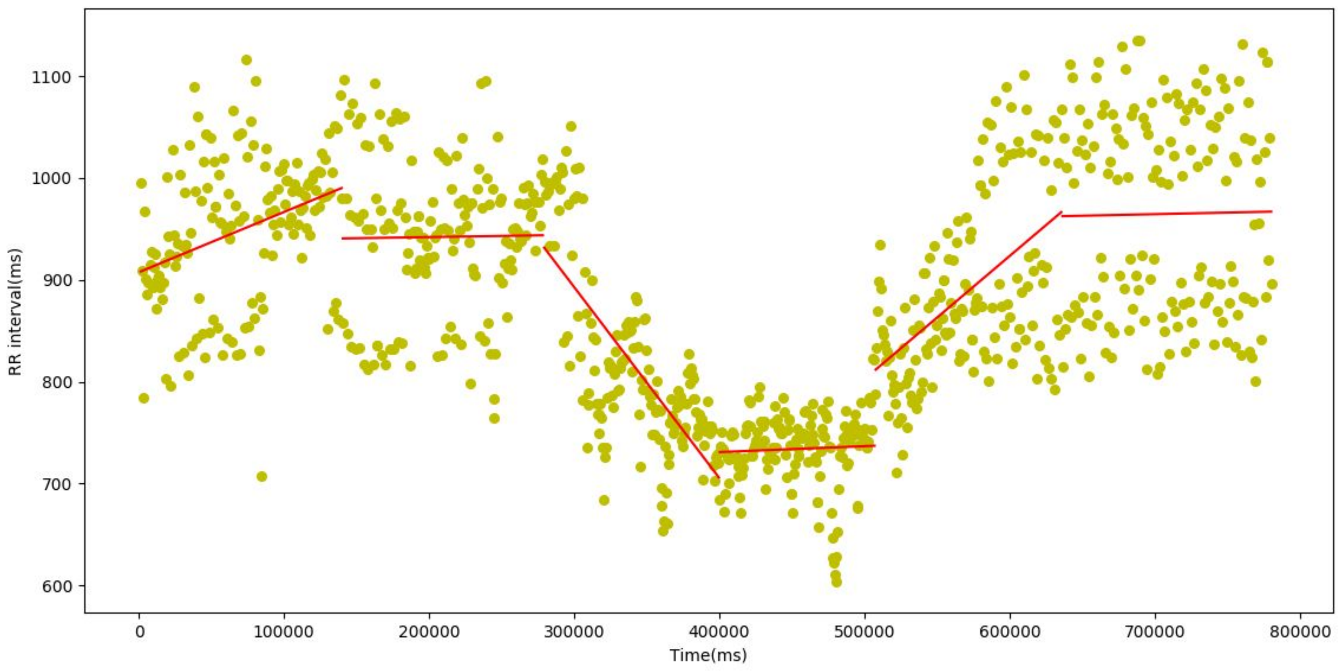
For interval 100



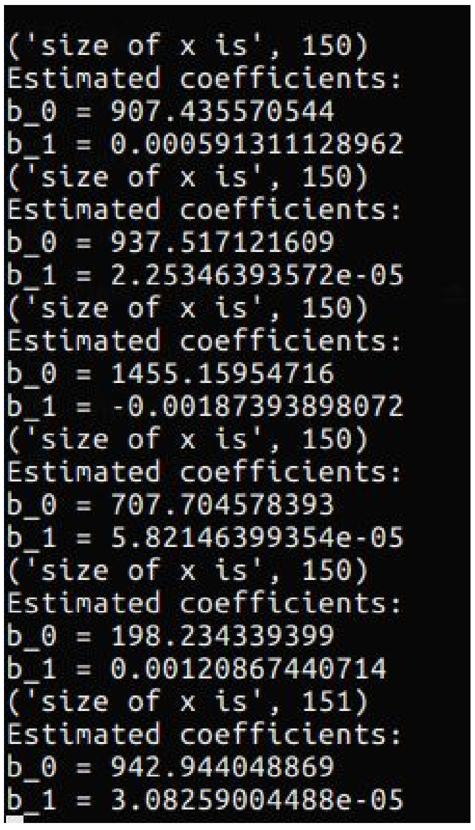
```
('size of x is', 100)
Estimated coefficients:
b_0 = 907.288885929
b_1 = 0.000578594239733
('size of x is', 100)
Estimated coefficients:
b_0 = 1028.8821934
b_1 = -0.000478436515916
('size of x is', 100)
Estimated coefficients:
b_0 = 867.635817492
b_1 = 0.000311728805532
('size of x is', 100)
Estimated coefficients:
b_0 = 1746.88769793
b_1 = -0.00280134107001
('size of x is', 100)
Estimated coefficients:
b_0 = 914.381864486
b_1 = -0.000424268460718
```

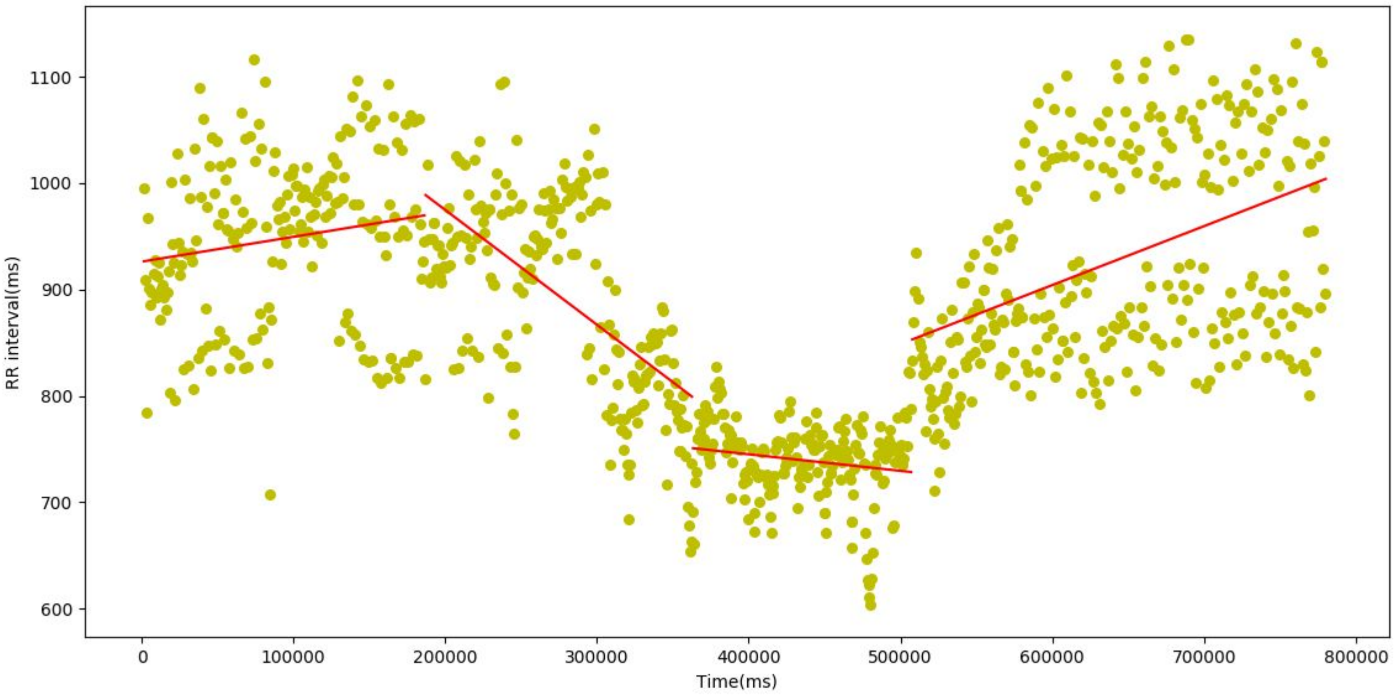


```
('size of x is', 100)
Estimated coefficients:
b_0 = 702.993300081
b_1 = 6.66456272965e-05
('size of x is', 100)
Estimated coefficients:
b_0 = 26.846466363
b_1 = 0.00152218914072
('size of x is', 100)
Estimated coefficients:
b_0 = 629.610148024
b_1 = 0.000505182172226
('size of x is', 101)
Estimated coefficients:
b_0 = 987.178784142
b_1 = -3.02838429799e-05
```

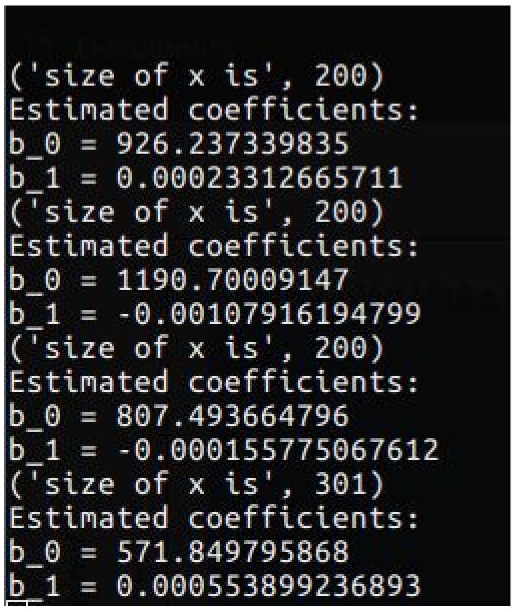


For interval 150

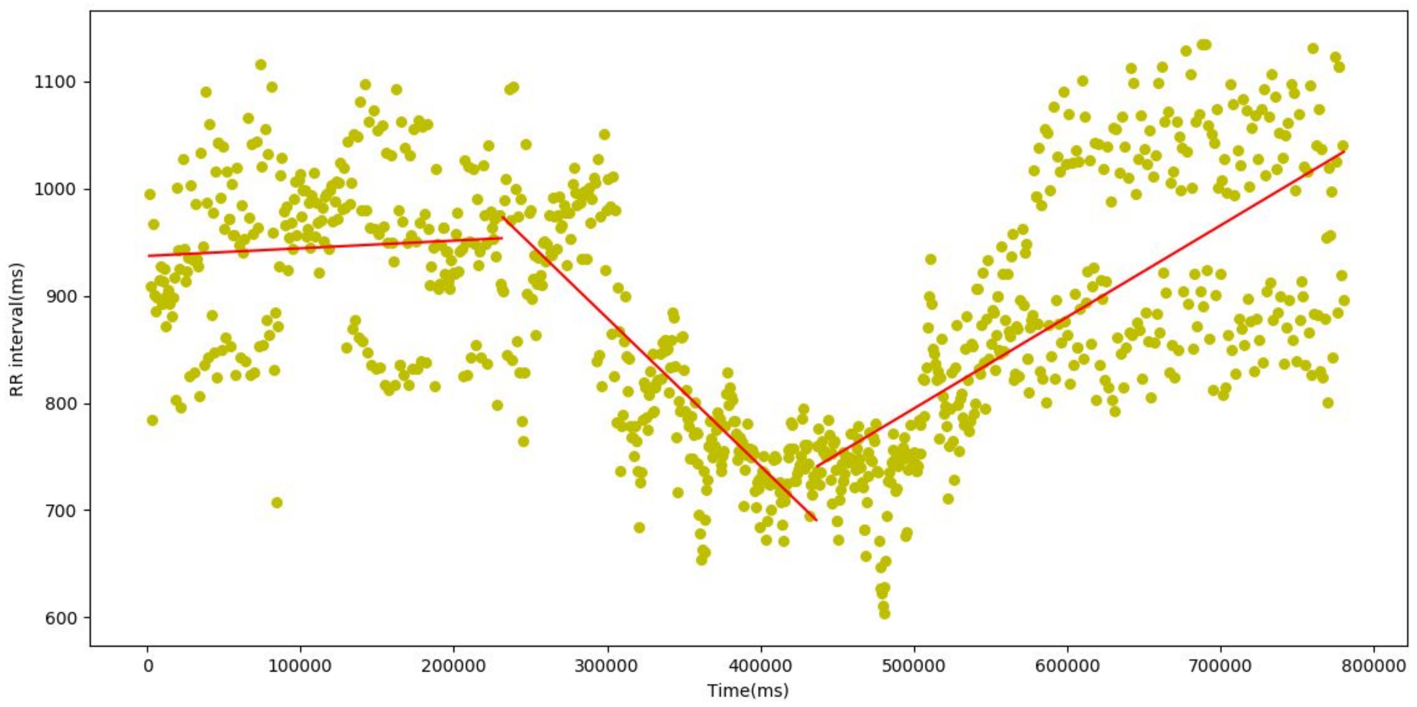

```
('size of x is', 150)
Estimated coefficients:
b_0 = 907.435570544
b_1 = 0.000591311128962
('size of x is', 150)
Estimated coefficients:
b_0 = 937.517121609
b_1 = 2.25346393572e-05
('size of x is', 150)
Estimated coefficients:
b_0 = 1455.15954716
b_1 = -0.00187393898072
('size of x is', 150)
Estimated coefficients:
b_0 = 707.704578393
b_1 = 5.82146399354e-05
('size of x is', 150)
Estimated coefficients:
b_0 = 198.234339399
b_1 = 0.00120867440714
('size of x is', 151)
Estimated coefficients:
b_0 = 942.944048869
b_1 = 3.08259004488e-05
```



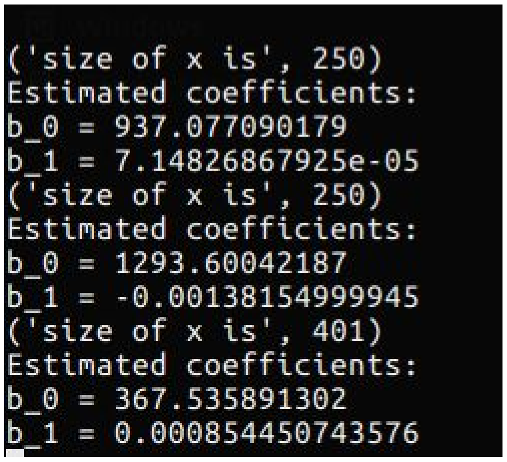
For interval 200



```
('size of x is', 200)
Estimated coefficients:
b_0 = 926.237339835
b_1 = 0.00023312665711
('size of x is', 200)
Estimated coefficients:
b_0 = 1190.70009147
b_1 = -0.00107916194799
('size of x is', 200)
Estimated coefficients:
b_0 = 807.493664796
b_1 = -0.000155775067612
('size of x is', 301)
Estimated coefficients:
b_0 = 571.849795868
b_1 = 0.000553899236893
```



For Interval 250



```
('size of x is', 250)
Estimated coefficients:
b_0 = 937.077090179
b_1 = 7.14826867925e-05
('size of x is', 250)
Estimated coefficients:
b_0 = 1293.60042187
b_1 = -0.00138154999945
('size of x is', 401)
Estimated coefficients:
b_0 = 367.535891302
b_1 = 0.000854450743576
```

Conclusion :

As Linear Regression is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables. It is mostly used for finding out the relationship between variables and forecasting.

$$y = \theta_1 + \theta_2 . x$$

X → input data [In our implementation X is "Time" which is an independent variable]

Y → labels to data [In our Implementation Y is " R-R Interval "] .

Θ_1 → intercept

Θ_2 → coefficient of X

We found the relation between R-R interval and Time as the equation : RR (T) = b_0 + b_1*t ------------>(1)

For Single and Multiple intervals by evaluating different values of b_0 and b_1 based on the input given

References :

1.) www.docs.python.org

2.) www.pypi.org/project/bioread/

3.) Heart rate variability Document :

Standards of measurement, physiological interpretation, and clinical use Task Force of The European Society of Cardiology and The North America Society of Pacing and Electrophysiology.

4.) https://en.wikipedia.org/wiki/Electrocardiography

5.)https://www.geeksforgeeks.org/ml-linear-regression/