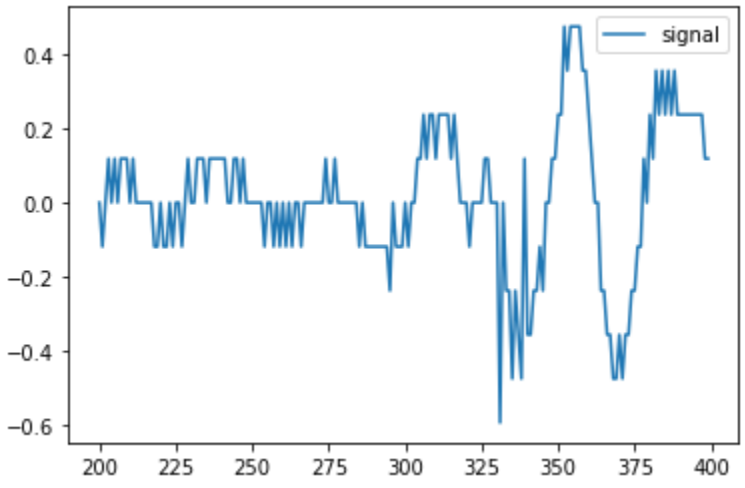
By - **Pranshu Rastogi**

Problem Assigned for the task
Detect Snoring from ballistocardiography signals provided

My approach is simple for the problem. I don't know that if this problem is solved in that way but my approach is very easy to implement and statistically proved too

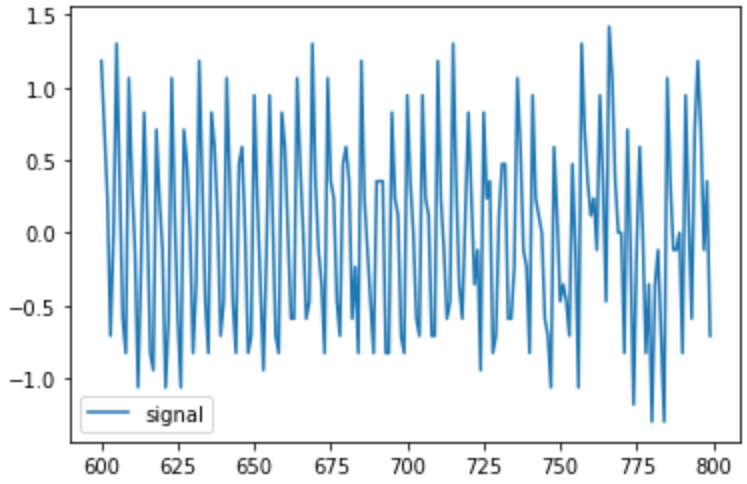
1. Do the standard scaling of the signals(preliminary step)
2. Stationarity of the series (important step)

Now the difference we see between snoring and not snoring purposes




signal with no snoring

fig1



Signal with snoring fig2

You can see the clear difference the snoring one has many fluctuations in comparison to the not snoring one

Now statistically speaking here is the proof that the snoring and not snoring data is significantly different

```
H0 = Null hypothesis that the expected values(mean) of the population are
the same mean of pop1 - mean of pop2 = 0, show that there is no
significant difference between the populations
H1 = conjugate of the H0 also called the alternative hypothesis
```

we have some populations samples so we can do some t-test and sample testing like a 2-tailed test for the population so that we can say that the data is significantly difference

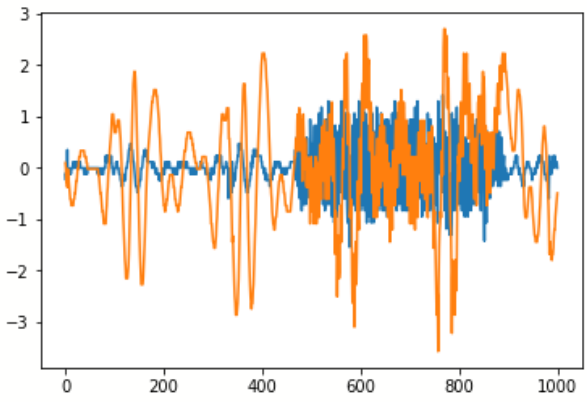


fig 3

You can see that the stationarity of the data is very good at segregating the signals or snoring and not snoring

```
def detect_snor(df_og, faint_level = 0.43):

"This is the main function the process is simple first I have scaled and
stationaries the signal and then squared the signal some signals are then
suppressed because of faintness then applied the gradient on the suppressed
signals through which we can get the onset and offset of the snoring
input = raw data frame
faint_level = level of faintness the more it is the more it suppresses
return = the onset and offsets in seconds of the snoring
"
  df_og = standard_scaler(df_og)
  df = df_og.copy()
  df1 = diff(df.values)
  df['signal'] = pd.Series(df1)
  df = np.square(df)
  l = [] , v = [] ,time_slab = []

  for i in range(0,len(df)):
    if df.values.flatten()[i]>faint_level:
      l.append(i)
  bin_arr = 1*(np.gradient(l)>100)
  bin_arr[0] = 1
  for i in range(len(bin_arr)):
    if bin_arr[i]==1:
      v.append(l[i])
  i = 0
  while i<=len(v)-2:
    time_slab.append([v[i],v[i+1]])
    i = i + 2

  time_slab = pd.DataFrame(time_slab)
  time_slab.columns = ['start_time_sec','end_time_sec']
  time_slab['start_time_sec'] = time_slab['start_time_sec']/250
  time_slab['end_time_sec'] = time_slab['end_time_sec']/250
  return time_slab
```

**This is the prediction function the main essence of my solution is the**

**`bin_arr = 1*(np.gradient(l)>100)` .So why I have used this is only because of fig3 as if a region of the signal *which has more fluctuation the more change* we can use to find the onset and offsets and where the grad is > 100 we can see that there is a change in the signal showing the onset and offset of snoring.**

Result:-

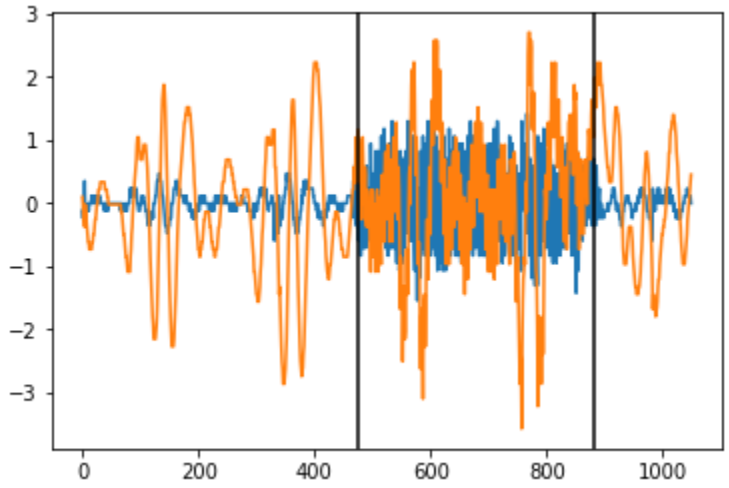


fig 4

The black lines are the starting and ending of the snoring per 1000 samples(4 seconds).
Hence this is it for the snoring part. I have exploited this in movement

## Movement part

Here we want to segment out every second where there is the movement detected through the signals we have 60000 samples of 4 minutes of data means we have 250 samples per second

Mostly everything is the same as more or less but a brief description

1. Standard scaling
2. Squaring the data
3. Essence of code

   scale_down(np.gradient(df_og.values.flatten())**2)

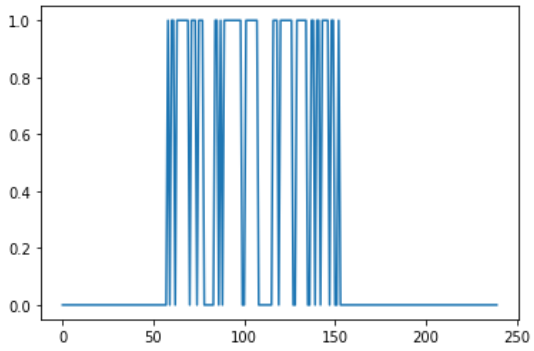
4. The parameter (‘`threshold_no_spikes`’) is very important because it is the threshold on and above which the second is termed as a movement
   I have used 3 movement spikes per second as a threshold meaning if data sample of 250 (1 second) has less than 3 movement spikes than that is not labeled as the movement one

```
def scale_down(series):
```

```
  series = (series - min(series)) / max(series)
  return series

def find_mov_per_sec(df_og, threshold_no_spikes = 3):
  """
  the procedure is same here i have squared the gradient with same suppression
  technique
  input_data = raw data frame
  threshold_no_spikes = number of spikes per second below this is not considered
  output = list showing whether the second has movement or not
  """
  df_og = standard_scaler(df_og)
  df_og = np.square(df_og)
  values = scale_down(np.gradient(df_og.values.flatten())**2)
  values = 1*(values>0.2)
  out = []
  for i in range(0 , len(values), 250):
    if np.sum(values[i:i+250])>=threshold_no_spikes:
      out.append(1)
    else:
      out.append(0)
  plt.plot(out)
  return out
time_slab_per_sec = find_mov_per_sec(df_og)
```

The data is also statistically different from the movement and the not movement population by the same two-tailed t-test with same hypothesis explained above
Here are some results for taskfile1.csv here the spikes shows that there is a movement in a second the scale is on seconds showing 240 seconds(4 minutes)



**Conclusion -**

**Thanks for the opportunity for an interesting and very good problem. It makes me think a lot and also makes me think out of the box or rather conventional way. I have tried to statistically justify the data and not used any ML/DL algorithm which i took as a challenge and it was really really great for me to do the project and before starting it i didn't even knew that i will came up to this solution**

**Thanks Dozee for the Assignment**
**I also want to apologize for my very late submission.**