Midterm Polygraph Assignment

Psychophysiological Engineering

Wesley Liao

2018-11-02

Table of Contents

[Background 2](#_Toc528972036)

[Approach 2](#_Toc528972037)

[Signals and Sensors 3](#_Toc528972038)

[Electrodermal Activity 3](#_Toc528972039)

[Systolic Pressure 4](#_Toc528972040)

[Respiratory RMS 5](#_Toc528972041)

[Inter-Beat-Interval 5](#_Toc528972042)

[Tests 6](#_Toc528972043)

[Test Sensitivity 7](#_Toc528972044)

[Weighting of Tests in Final Determination 9](#_Toc528972045)

[Results: 11](#_Toc528972046)

[Discussion 13](#_Toc528972047)

[Code: 14](#_Toc528972048)

## Background

Selected Papers:

Podlesny, J. A., & Raskin, D. C. (1977). Physiological measures and the detection of deception. *Psychological Bulletin*, *84*(4), 782–799. https://doi.org/10.1037/0033-2909.84.4.782

Dalgleish, T., Williams, J. M. G. ., Golden, A.-M. J., Perkins, N., Barrett, L. F., Barnard, P. J., … Watkins, E. (2007). *Detecting Concealed Information and Deception*. *Journal of Experimental Psychology: General* (Vol. 136).

Podlesny & Raskin was used as a general overview of common polygraph measures and in determining which signals would be most useful in detecting deception. It provided a wide range of information on all of the sensors used in polygraphs up until the late 70s.

Dalgleish et. al. was selected because of its inclusion of the timing of physiological responses after question events in a similar experiment. Only the sections on background and physiological responses were used. This was also selected as a source because it is published much more recently, providing updated information into the field of deception detection.

## Approach

Of the questions asked, only the questions directly asking if values of cash or check were removed from either the mailbox or the bag were analyzed. The “baseline” questions, e.g. “Have you met Dr. Lewis?”) were excluded because they were not relevant to the crime. The questions leading up to the crime (“Did you open envelope X”) were excluded because they could be easily confused with the several other similar envelopes or actions and are only indirectly related to the crime committed.

Constructing a matrix of the selected questions below based on each question’s relevance to the parameters, we create 6 question sets:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Cash | | | | Check | | | |
|  | Any | $20.00 | $41.00 | $470.16 | Any | $20.00 | $41.00 | $470.16 |
| Mailbox | 4B | 5B | 6B | 7B | 8B | 9B | 1C | 2C |
| Bag | 8C | 9C | 1D | 2D | 3D | 4D | 5D | 6D |

Question Sets:

1. "Mailbox": ["4B","5B","6B","7B","8B","9B","1C","2C"],
2. "Bag": ["8C","9C","1D","2D","3D","4D","5D","6D"],
3. "Cash": ["4B","5B","6B","7B","8C","9C","1D","2D"],
4. "Check": ["8B","9B","1C","2C","3D","4D","5D","6D"],
5. "$41": ["6B", "1D", "1C", "5D"],
6. "$470.16": ["7B","2D","2C","6D"]

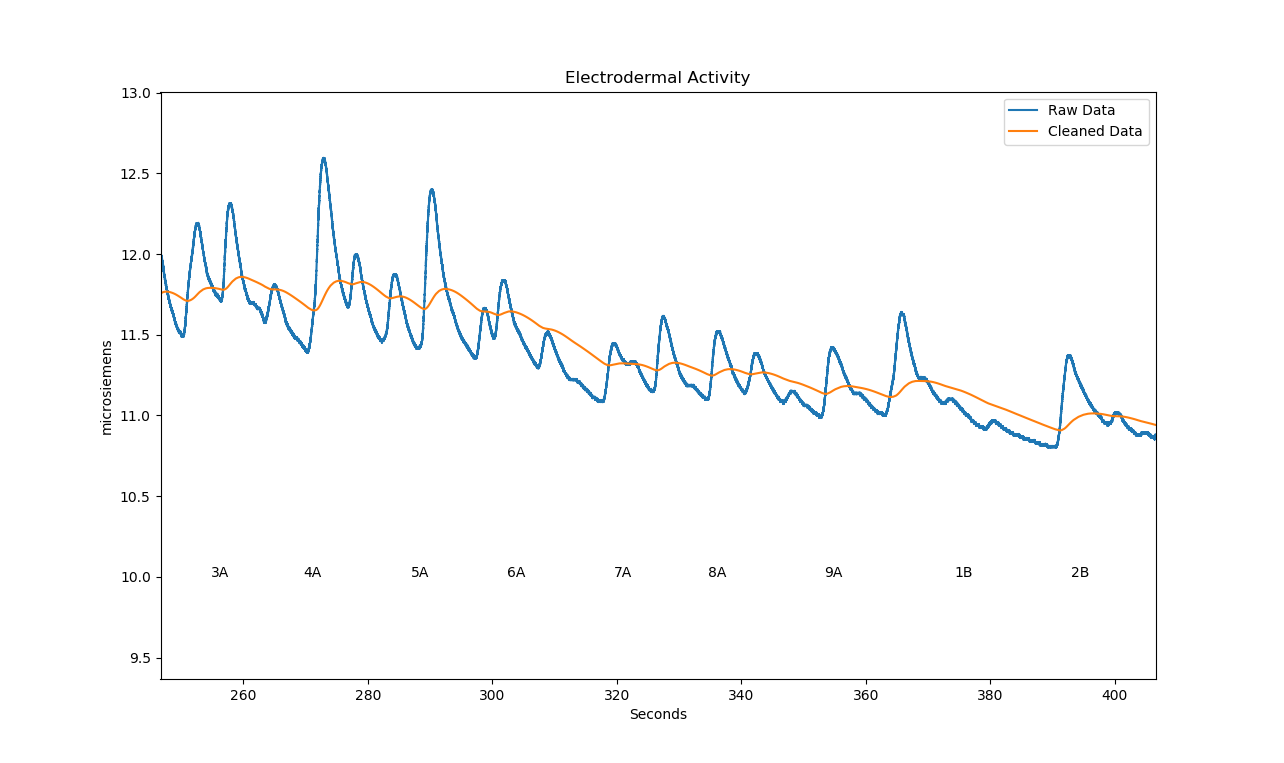
A set for $20 was not created because we know that the value of the cash was $41 and the value of the Check was $470.16. Each subject’s responses are tested for each of the 16 questions. Then to evaluate guilt, we compare each subject’s response to the “Mailbox” questions versus the “Bag” questions (and Cash vs Check, etc.) and see which response is greater. Subjects were not compared to each other. Subjects were only compared to themselves to avoid normalizing responses between different subjects.

## Signals

Four signals were used in detecting physiological responses. This section will cover the processing of each signal, from original data to final parameter.

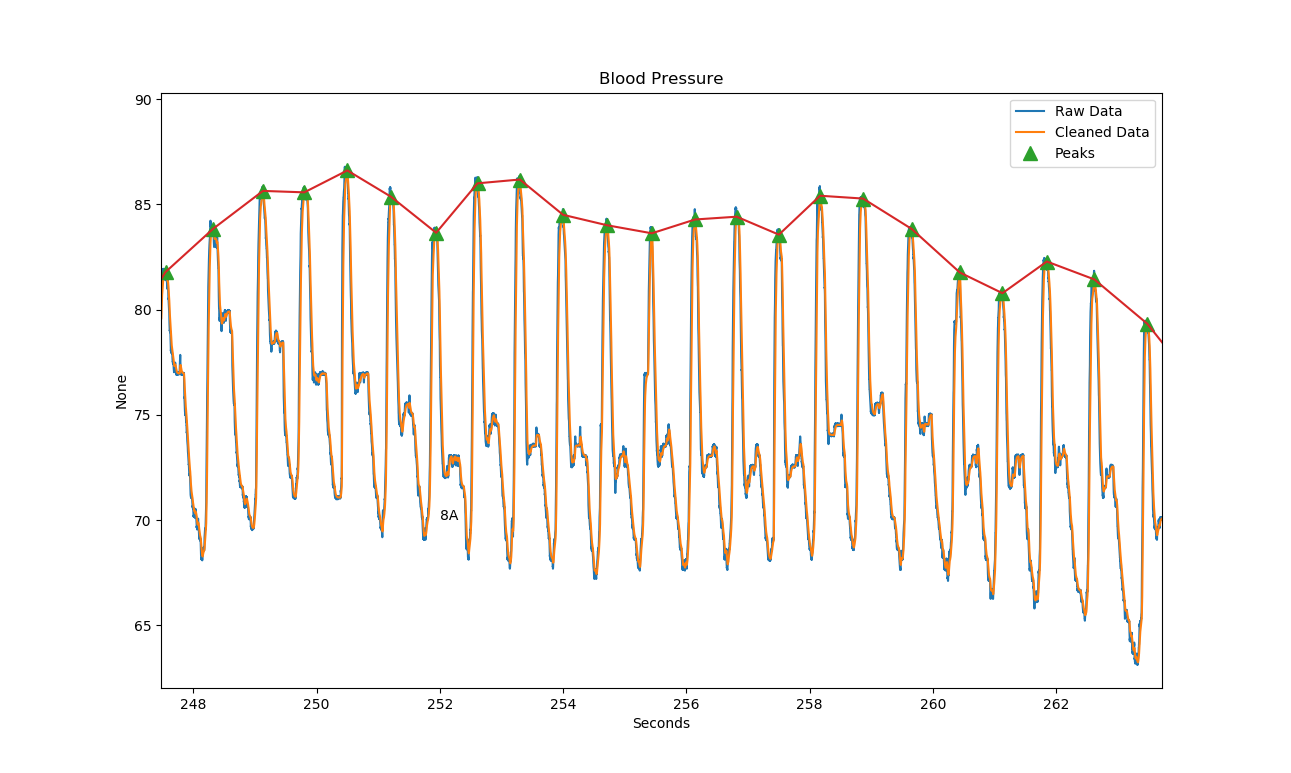
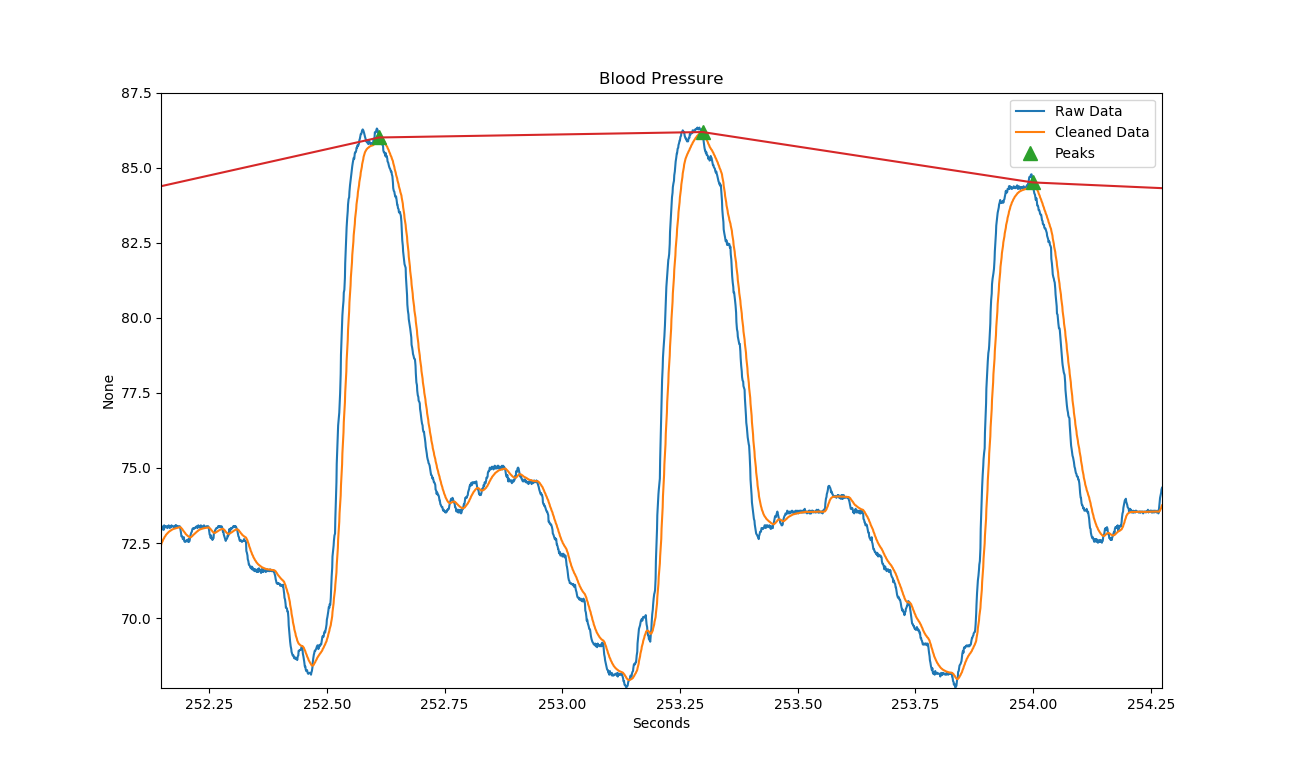
### Electrodermal Activity

From the original signal, an exponential decay weighted average was applied to smooth the signal.

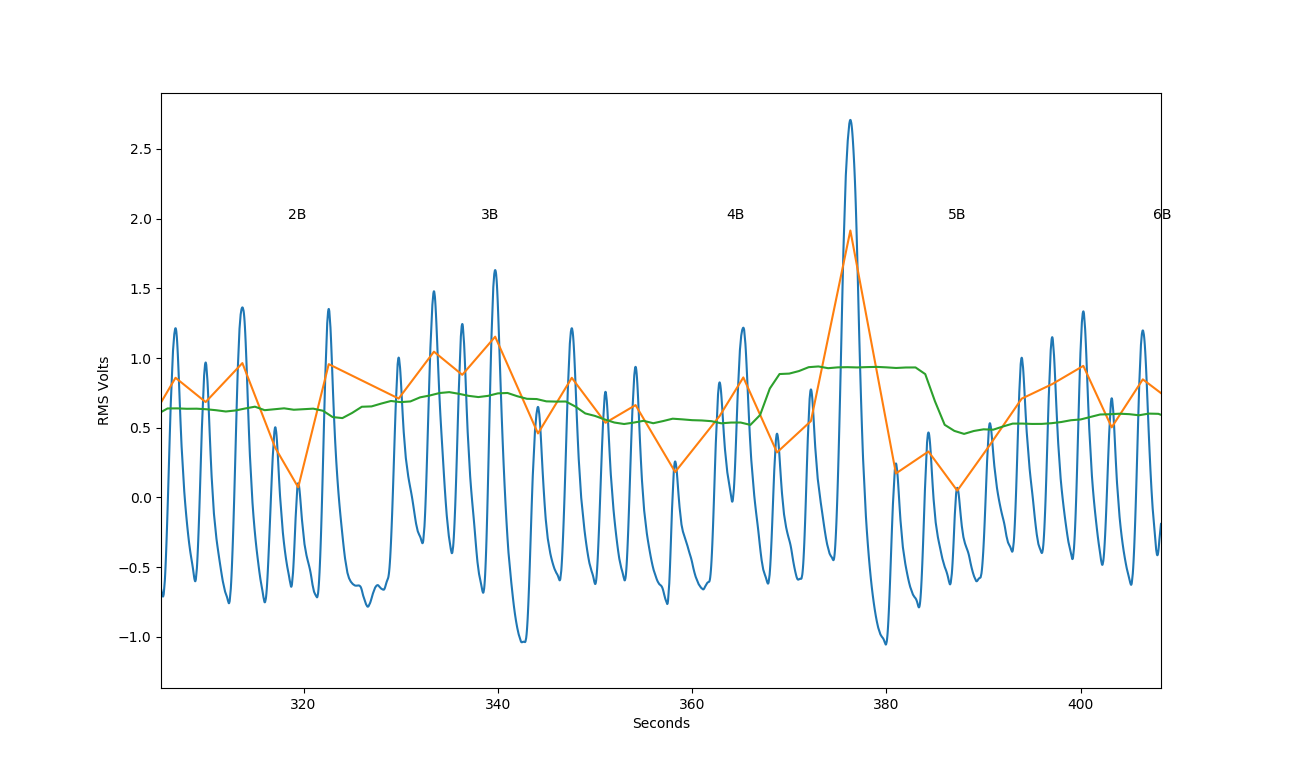


### Systolic Pressure

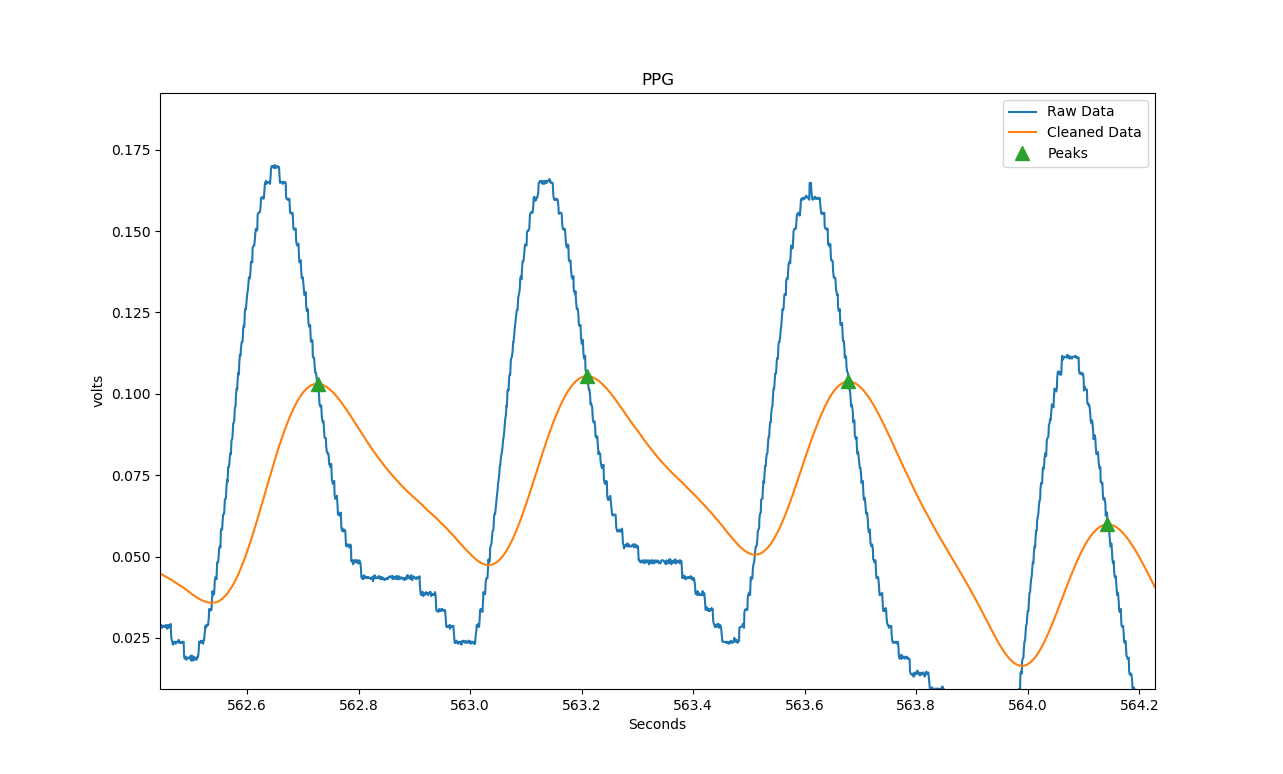
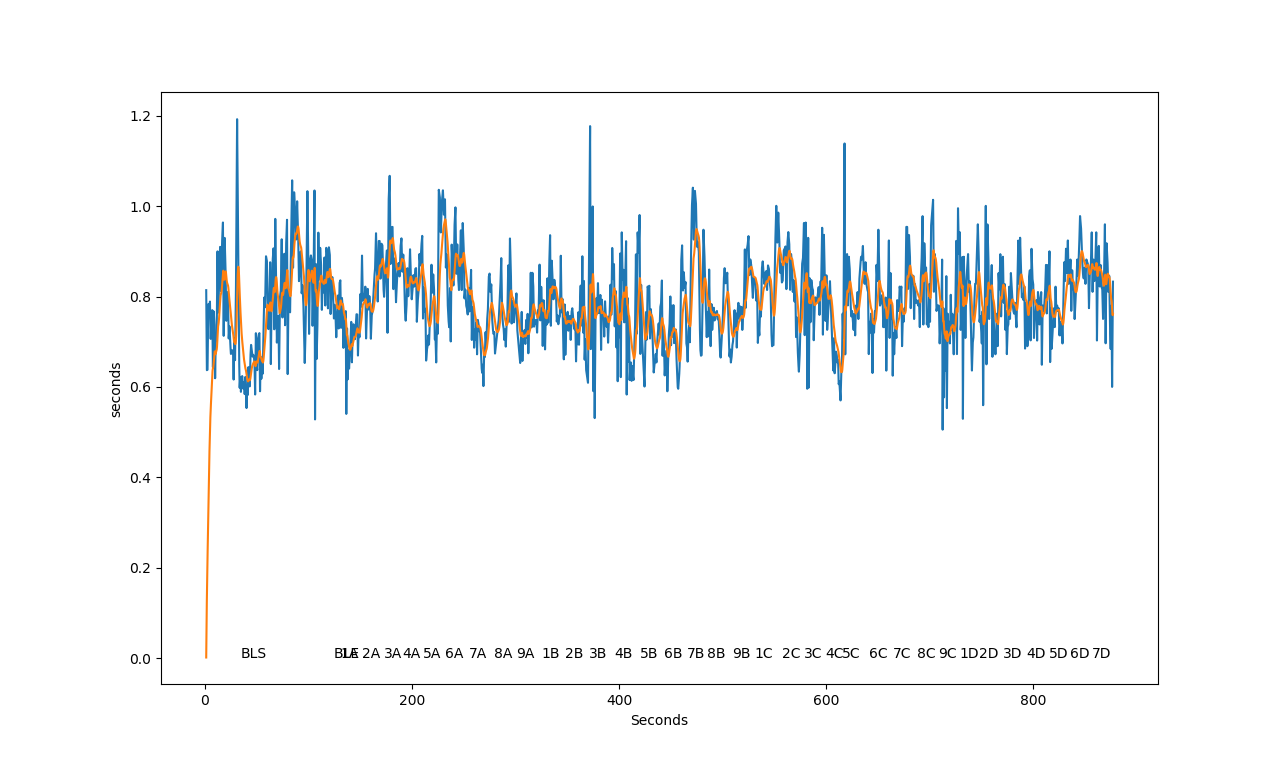
From the blood pressure signal, first a low-pass filter is applied to smooth out small variations in the signal, making peak-finding easier. The peaks are then found using the scipy find\_peaks function, and the peak values interpolated for systolic pressure.



### Respiratory RMS

The respiration signal was first smoothed with a low-pass filter, and then a spectrogram was generated form the data. The values of the spectrogram at each time window were summed and then the square root of the sum was found. This value estimates the RMS of the respiration. Plotted below is the filtered signal (blue), compared with the RMS estimate (green), and the peak value divided by the square root of 2, another crude estimate for RMS. 

### Inter-Beat-Interval

From the PPG signal, because we only care about the peak spacing, and not the value, the data is heavily filtered to provide the clearest peaks. The IBI is generated from the difference in times between each peak of the filtered PPG signal. The filtered signal and peaks are shown below. The IBI signal is also low-pass filtered to smooth inconsistencies.

## Tests

To determine when a physiological response was significant, tests are used to analyze the parameters after each pertinent question and produce a score for each subject on each question. Several tests were developed:

* pre- and post-question
  + mean, sum, and min/max comparisons
* baseline to post-question
  + mean, sum, and min/max comparisons
* threshold Z-score
  + mean, min/max, greater/less than threshold

All of these tests were tried at some points, but threshold Z-score tests were the only tests used in the final determination. After a delay following each relevant question, a segment of each parameter was considered for the test. For the threshold Z-score tests used, the mean value of the Z-score during this segment was recorded and compared to a threshold.

Because the question markers are at the beginning of each question, 3 seconds was given for each question to be asked.

quesdelay = 3

In addition to the delay for the question, depending on the type of signal being analyzed, an additional delay was added to wait for the response to manifest in the subject, and then the period of time to be analyzed was varied, according to the sources.

eda\_delay = 1

eda\_period = 8

syspress\_delay = 3

syspress\_period = 11

resprms\_delay = 1

resprms\_period = 20

ibi\_delay = 4

ibi\_period = 10

Each test has been set up to provide a score of 1 when a question has a significant physiological response and a 0 otherwise. The Z-score thresholds were adjusted so that each test would have about the same sensitivity to responses. The final thresholds are shown below:

eda\_zthresh = 0.4

syspress\_zthresh = -0.5

resprms\_zthresh = -0.45

ibi\_zthresh = -0.1

The sensitivity is found by taking the sums of all responses from each subject and averaging them. Below is the response data for all subjects and questions for each test:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Threshold Pos EDA Z-Score | | |  |  |  |  |  |  |  |  |  |  |  |  |  | Average: | 6.714286 |
|  | 4B | 1D | 8C | 3D | 2C | 5D | 5B | 6D | 9C | 6B | 2D | 4D | 7B | 9B | 1C | 8B | Sum |
| Subject A | 1 | 0 | 1 | 1 |  | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 13 |
| Subject B | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 2 |
| Subject C | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 7 |
| Subject D | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 11 |
| Subject E | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 5 |
| Subject F | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 6 |
| Subject G | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 3 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Threshold Pos SysPress Z-Score | | | |  |  |  |  |  |  |  |  |  |  |  |  | Average: | 6.571429 |
|  | 4B | 1D | 8C | 3D | 2C | 5D | 5B | 6D | 9C | 6B | 2D | 4D | 7B | 9B | 1C | 8B | Sum |
| Subject A | 1 | 0 | 0 | 0 |  | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 6 |
| Subject B | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| Subject C | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| Subject D | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 6 |
| Subject E | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 8 |
| Subject F | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 15 |
| Subject G | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 7 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Threshold Neg RespRms Z-Score | | | |  |  |  |  |  |  |  |  |  |  |  |  | Average: | 6.571429 |
|  | 4B | 1D | 8C | 3D | 2C | 5D | 5B | 6D | 9C | 6B | 2D | 4D | 7B | 9B | 1C | 8B | Sum |
| Subject A | 0 | 1 | 0 | 0 |  | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 6 |
| Subject B | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 8 |
| Subject C | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Subject D | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 11 |
| Subject E | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 5 |
| Subject F | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 6 |
| Subject G | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 9 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Threshold Neg IBI Z-Score | | |  |  |  |  |  |  |  |  |  |  |  |  |  | Average: | 6.857143 |
|  | 4B | 1D | 8C | 3D | 2C | 5D | 5B | 6D | 9C | 6B | 2D | 4D | 7B | 9B | 1C | 8B | Sum |
| Subject A | 0 | 0 | 1 | 0 |  | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| Subject B | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 9 |
| Subject C | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 7 |
| Subject D | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 9 |
| Subject E | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 5 |
| Subject F | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 9 |
| Subject G | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 7 |

## Weighting of Tests in Final Determination

With each of our test sensitivities adjusted, we now decide how to combine the data from each test.

We were given the distribution of thieves within the main categories:

There were 3 check and 4 cash thieves.

There were 4 office and 3 mail room thieves.

Looking at each test’s results, and comparing to the known distribution, we can rank each test by how accurate they appear to be:

From Threshold Pos EDA Z-Score test results:

Subject A stole a Check for $470.16 from Bag

Subject B stole $41 in Cash from Mailbox

Subject C stole $41 in Cash from Mailbox

Subject D stole a Check for $470.16 from Mailbox

Subject E stole a Check for $470.16 from Bag

Subject F stole a Check for $470.16 from Mailbox

Subject G stole a Check for $470.16 from Mailbox

(4 known incorrect)

From Threshold Pos SysPress Z-Score test results:

Subject A stole $41 in Cash from Mailbox

Subject B stole a Check for $470.16 from Bag

Subject C stole $41 in Cash from Mailbox

Subject D stole $41 in Cash from Mailbox

Subject E stole $41 in Cash from Bag

Subject F stole $41 in Cash from Bag

Subject G stole a Check for $470.16 from Bag

(1 known incorrect)

From Threshold Neg RespRms Z-Score test results:

Subject A stole a Check for $470.16 from Bag

Subject B stole a Check for $470.16 from Bag

Subject C stole a Check for $470.16 from Bag

Subject D stole a Check for $470.16 from Mailbox

Subject E stole a Check for $470.16 from Bag

Subject F stole $41 in Cash from Mailbox

Subject G stole a Check for $470.16 from Bag

(4 known incorrect)

From Threshold Neg IBI Z-Score test results:

Subject A stole $41 in Cash from Bag

Subject B stole $41 in Cash from Bag

Subject C stole $41 in Cash from Mailbox

Subject D stole a Check for $470.16 from Mailbox

Subject E stole a Check for $470.16 from Mailbox

Subject F stole a Check for $470.16 from Bag

Subject G stole $41 in Cash from Bag

(0 known incorrect)

Ranking our tests in order of apparent accuracy, we have IBI, then Systolic Pressure, then a tie for EDA and Respiratory RMS. Because it is easy for subjects to manipulate their breathing, but have most likely not trained to control their EDA, EDA is ranked above Respiratory RMS.

Final weights:

Threshold Pos EDA Z-Score 0.5

Threshold Pos SysPress Z-Score 0.75

Threshold Neg RespRms Z-Score 0.25

Threshold Neg IBI Z-Score 1.0

## Results:

Our final results are as follows:

* Subject A stole $41 Cash from Mailbox
* Subject B stole $41 Cash from Bag
* Subject C stole $41 Cash from Mailbox
* Subject D stole a Check for $470.16 from Mailbox
* Subject E stole a Check for $470.16 from Bag
* Subject F stole a Check for $470.16 from Bag
* Subject G stole $41 Cash from Bag

The calculation is a simple comparison, if the total scores from the mailbox events are greater than the bag events, and similarly the combination of $41 dollar events and cash events compared to the check and $470.16 events. The final data from the tests and final aggregate sum is shown in tables below:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| == Question Summed Test Results == | | | |  |  |  |
|  |  |  |  |  |  |  |
| Threshold Pos EDA Z-Score | | |  |  |  |  |
|  | Mailbox | Bag | Cash | Check | $41 | $470.16 |
| Subject A | 4.5 | 5.25 | 5.25 | 4.5 | 1.5 | 2.25 |
| Subject B | 1.5 | 0 | 1.5 | 0 | 0.75 | 0.75 |
| Subject C | 3 | 2.25 | 4.5 | 0.75 | 1.5 | 1.5 |
| Subject D | 5.25 | 3 | 3 | 5.25 | 2.25 | 2.25 |
| Subject E | 0.75 | 3 | 0.75 | 3 | 0.75 | 1.5 |
| Subject F | 3.75 | 0.75 | 1.5 | 3 | 0.75 | 2.25 |
| Subject G | 2.25 | 0 | 1.5 | 0.75 | 0 | 0.75 |
|  |  |  |  |  |  |  |
| Threshold Pos SysPress Z-Score | | | |  |  |  |
|  | Mailbox | Bag | Cash | Check | $41 | $470.16 |
| Subject A | 3 | 0 | 2 | 1 | 0.5 | 0.5 |
| Subject B | 0.5 | 0.5 | 0 | 1 | 0.5 | 0.5 |
| Subject C | 1 | 0 | 1 | 0 | 0 | 0 |
| Subject D | 2.5 | 0.5 | 2.5 | 0.5 | 0.5 | 0.5 |
| Subject E | 1.5 | 2.5 | 3 | 1 | 1 | 0 |
| Subject F | 3.5 | 4 | 4 | 3.5 | 2 | 2 |
| Subject G | 1 | 2.5 | 1.5 | 2 | 0.5 | 1 |
|  |  |  |  |  |  |  |
| Threshold Neg RespRms Z-Score | | | |  |  |  |
|  | Mailbox | Bag | Cash | Check | $41 | $470.16 |
| Subject A | 0.75 | 0.75 | 0.75 | 0.75 | 0.25 | 0.25 |
| Subject B | 0.75 | 1.25 | 0.75 | 1.25 | 0.5 | 0.5 |
| Subject C | 0 | 0.25 | 0 | 0.25 | 0 | 0.25 |
| Subject D | 1.5 | 1.25 | 1 | 1.75 | 0.75 | 0.5 |
| Subject E | 0.25 | 1 | 0.5 | 0.75 | 0 | 0.75 |
| Subject F | 1.5 | 0 | 0.75 | 0.75 | 0.5 | 0.25 |
| Subject G | 0.5 | 1.75 | 0.75 | 1.5 | 0.5 | 0.75 |
|  |  |  |  |  |  |  |
| Threshold Neg IBI Z-Score | | |  |  |  |  |
|  | Mailbox | Bag | Cash | Check | $41 | $470.16 |
| Subject A | 1 | 1 | 2 | 0 | 1 | 0 |
| Subject B | 3 | 6 | 6 | 3 | 2 | 2 |
| Subject C | 4 | 3 | 4 | 3 | 2 | 0 |
| Subject D | 5 | 4 | 4 | 5 | 1 | 2 |
| Subject E | 3 | 2 | 1 | 4 | 0 | 1 |
| Subject F | 3 | 6 | 3 | 6 | 3 | 2 |
| Subject G | 3 | 4 | 5 | 2 | 2 | 2 |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| == All Test Aggregate Results == | | | |  |  |  |
|  |  |  |  |  |  |  |
| Threshold Neg IBI Z-Score | | |  |  |  |  |
|  | Mailbox | Bag | Cash | Check | $41 | $470.16 |
| Subject A | 9.25 | 7 | 10 | 6.25 | 3.25 | 3 |
| Subject B | 5.75 | 7.75 | 8.25 | 5.25 | 3.75 | 3.75 |
| Subject C | 8 | 5.5 | 9.5 | 4 | 3.5 | 1.75 |
| Subject D | 14.25 | 8.75 | 10.5 | 12.5 | 4.5 | 5.25 |
| Subject E | 5.5 | 8.5 | 5.25 | 8.75 | 1.75 | 3.25 |
| Subject F | 11.75 | 10.75 | 9.25 | 13.25 | 6.25 | 6.5 |
| Subject G | 6.75 | 8.25 | 8.75 | 6.25 | 3 | 4.5 |

## Discussion

My approach uses only systematic comparisons of the data to determine which category is more likely, removing any human error or bias in interpreting the results. However, by removing the human element, the computer system is more vulnerable to artifacts in the data, which could be more easily recognized by a human analyzer and ignored. My hope is that the consistency gained by computer analysis outweighs the risk of false positives because of artifacts.

By combining several parameters and many redundant questions, I hope that the variation can be mitigated. Artifacts in the data, inconsistent responses, unknown environmental factors, should be “averaged out.” The parameters should be independent when considering disturbances but are not independent when physiological changes are occurring. This should help the overall accuracy of the tests.

The results here seem reasonable, however adjustments in the test weights, the window of signal analyzed after each question, the thresholds of the tests, all affect the results, and I do not have a concrete way to validate if the methods are good.

In my approach, I analyzed equal numbers of questions on both sides of the two decisions, all the parameters are converted to Z-scores, which effectively normalizes them, and I adjusted the Z-score thresholds so each parameter test all had similar sensitivities. Then the scores are simply added, weighted in order of how reliable each measure appeared to be. My approach may not be optimal, but it should be at least fair across the subjects, and if all subjects go through the same tests, their own differences in responses to the questions should show through.

## Code:

import numpy as np

from scipy import fftpack, signal, interpolate, stats

import matplotlib.pyplot as plt

import csv

##############################################################################

# Section copied from A. Coady posted on stackoverflow, accessed 2018.10.29:

# https://stackoverflow.com/questions/268272/getting-key-with-maximum-value-in-dictionary#280156

def keywithmaxval(d):

""" a) create a list of the dict's keys and values;

b) return the key with the max value"""

v=list(d.values())

k=list(d.keys())

return k[v.index(max(v))]

# End of copied section

###############################################################################

###############################################################################

# Filters

#

def lowpass(data, cutoff\_Hz, order):

nyquist\_Hz = data.samplerate\_Hz/2

Wn = (cutoff\_Hz/data.samplerate\_Hz)/nyquist\_Hz

b, a = signal.butter(order, Wn, btype='low', analog=False)

filtereddata = signal.lfilter(b, a, data.values)

return Timeseries(filtereddata, timestamps=data.times, samplerate\_Hz=data.samplerate\_Hz, units=data.units)

def notch(data, notch\_Hz, bandwidth):

nyquist\_Hz = data.samplerate\_Hz/2

w0 = notch\_Hz/nyquist\_Hz

quality = w0/bandwidth

b, a = signal.iirnotch(w0, quality)

filtereddata = signal.lfilter(b, a, data.values)

return Timeseries(filtereddata, timestamps=data.times, samplerate\_Hz=data.samplerate\_Hz, units=data.units)

#

###############################################################################

###############################################################################

# Core Classes

#

class Timeseries:

def \_\_init\_\_(self, datapoints, timestamps=None, samplerate\_Hz=1.0, starttime=0.0, units=None):

self.values = datapoints

self.units = units

self.samplerate\_Hz = samplerate\_Hz

if timestamps is not None and (len(datapoints) == len(timestamps)):

self.times = timestamps

else:

self.times = np.arange(starttime, samplerate\_Hz\*len(datapoints), samplerate\_Hz)

self.interp = interpolate.interp1d(self.times, self.values)

def plot(self):

plt.plot(self.times, self.values)

plt.xlabel("Seconds")

plt.ylabel(str(self.units))

class SensorData:

def \_\_init\_\_(self, name, rawdata):

self.name = name

self.rawdata = rawdata

print(" - Cleaning data")

self.cleandata = self.process(self.rawdata)

def process(self, data):

return data

def plot(self):

self.rawdata.plot()

self.cleandata.plot()

plt.legend(["Raw Data", "Cleaned Data"])

plt.title(self.name)

class PeriodicSensorData(SensorData):

def \_\_init\_\_(self, name, rawdata):

super().\_\_init\_\_(name, rawdata)

# print(" - Processing FFT")

# self.fft = fftpack.fftshift(fftpack.rfft(self.cleandata.values))

# bins = len(self.fft)

# nyquist = self.cleandata.samplerate\_Hz/2

# freqmax = nyquist/2

# self.freqrange = np.linspace(-freqmax, freqmax, bins)

print(" - Finding Peaks")

self.peaks = self.find\_peaks(self.cleandata)

def plot(self):

super().plot()

plt.plot(self.peaks.times, self.peaks.values, linestyle="None", marker="^", markersize=10)

plt.legend(["Raw Data", "Cleaned Data", "Peaks"])

# def plotfft(self):

# plt.plot(self.freqrange, self.fft)

# plt.xlabel("Frequency (Hz)")

# plt.ylabel(str(self.cleandata.units))

def find\_peaks(self,data):

peaks, properties = signal.find\_peaks(data.values)

if len(peaks)>0:

return Timeseries(data.values[peaks], timestamps=data.times[peaks], samplerate\_Hz=None)

else:

return None

#

###############################################################################

###############################################################################

# Sensor Data Classes

#

class BloodPressure(PeriodicSensorData):

def \_\_init\_\_(self, data, times, peakinterval, lowbound):

self.peakinterval = peakinterval

self.lowbound = lowbound

rawdata = Timeseries(data, timestamps=times, samplerate\_Hz=1000, units="mmHg")

super().\_\_init\_\_("Blood Pressure", rawdata)

def process(self, data):

cutoff\_Hz = 12500

order = 1

return lowpass(data, cutoff\_Hz, order)

def find\_peaks(self,data):

peaks, properties = signal.find\_peaks(data.values, distance=self.peakinterval, height=self.lowbound)

if len(peaks)>0:

return Timeseries(data.values[peaks], timestamps=data.times[peaks], samplerate\_Hz=None)

else:

return None

class RSP(PeriodicSensorData):

def \_\_init\_\_(self, data, times):

rawdata = Timeseries(data, timestamps=times, samplerate\_Hz=1000, units="volts")

super().\_\_init\_\_("RSP", rawdata)

def process(self, data):

cutoff\_Hz = 100

order = 1

lp = lowpass(data, cutoff\_Hz, order)

notch\_Hz = 0.0001

bandwidth = 0.0001

return notch(lp, notch\_Hz, bandwidth)

def find\_peaks(self,data):

peaks, properties = signal.find\_peaks(data.values, height=0)

if len(peaks)>0:

return Timeseries(data.values[peaks], timestamps=data.times[peaks], samplerate\_Hz=None)

else:

return None

class PPG(PeriodicSensorData):

def \_\_init\_\_(self, data, times, peakinterval, lowbound):

self.peakinterval = peakinterval

self.lowbound = lowbound

rawdata = Timeseries(data, timestamps=times, samplerate\_Hz=1000, units="volts")

super().\_\_init\_\_("PPG", rawdata)

def process(self, data):

cutoff\_Hz = 1000

order = 1

return lowpass(data, cutoff\_Hz, order)

def find\_peaks(self,data):

peaks, properties = signal.find\_peaks(data.values, distance=self.peakinterval, height=self.lowbound)

if len(peaks)>0:

return Timeseries(data.values[peaks], timestamps=data.times[peaks], samplerate\_Hz=None)

else:

return None

class AvgBloodPressure(SensorData):

def \_\_init\_\_(self, data, times):

rawdata = Timeseries(data, timestamps=times, samplerate\_Hz=1000, units="mmHg")

super().\_\_init\_\_("Mean Arterial Pressure", rawdata)

class AvgPulse(SensorData):

def \_\_init\_\_(self, data, times):

rawdata = Timeseries(data, timestamps=times, samplerate\_Hz=1000, units="BPM")

super().\_\_init\_\_("Pulse", rawdata)

class ElectrodermalActivity(SensorData):

def \_\_init\_\_(self, data, times):

rawdata = Timeseries(data, timestamps=times, samplerate\_Hz=1000, units="microsiemens")

super().\_\_init\_\_("Electrodermal Activity", rawdata)

def process(self, data):

timestep = 0.001

timerange = np.arange(0,60,timestep)

decay = 10

weightingfunc = (np.exp(-timerange/decay)/decay)\*timestep

filtereddata = signal.fftconvolve(data.values, weightingfunc)[:len(data.values)]

notch\_Hz = 0.001

bandwidth = 0.001

# return notch(data, notch\_Hz, bandwidth)

return Timeseries(filtereddata, timestamps=data.times, samplerate\_Hz=data.samplerate\_Hz, units=data.units)

#

###############################################################################

###############################################################################

# Parameter Classes

#

class Parameter():

def \_\_init\_\_(self, name, sensor):

self.name = name

self.rawdata = self.extract(sensor)

self.cleandata = self.process(self.rawdata)

def extract(self, sensor):

return sensor.cleandata

def process(self, data):

return data

class SystolicPressure(Parameter):

def extract(self, sensor):

return Timeseries(sensor.peaks.values,

timestamps=sensor.peaks.times,

samplerate\_Hz=None,

units=sensor.peaks.units)

class IBI(Parameter):

def extract(self, sensor):

return Timeseries(np.diff(sensor.peaks.times),

timestamps=sensor.peaks.times[1:],

samplerate\_Hz=None,

units=sensor.peaks.units)

def process(self, data):

dropped = Timeseries(data.values[np.where(data.values<1.2)],

timestamps=data.times[np.where(data.values<1.2)],

samplerate\_Hz=None,

units=data.units)

timerange= np.arange(dropped.times[0], dropped.times[-1], 0.01)

resampled = Timeseries(dropped.interp(timerange), timerange, samplerate\_Hz=100, units="seconds")

self.rawdata = resampled

return lowpass(resampled, 5, 1)

class RespRMS(Parameter):

def extract(self, sensor):

sampleperiod = 0.001

fs = 1000

seconds = 20

overlap = 0.95

timerange = np.arange(sensor.cleandata.times[0], sensor.cleandata.times[-1], sampleperiod)

f, t, Sxx = signal.spectrogram(sensor.cleandata.interp(timerange),

fs,

nperseg=seconds\*fs,

noverlap=seconds\*fs\*overlap,

scaling='spectrum')

# plt.figure()

# plt.pcolormesh(Sxx)

# plt.ylim([0, seconds])

return Timeseries(np.sqrt(np.sum(Sxx, axis=0)),

timestamps=t,

samplerate\_Hz=fs,

units="RMS Volts")

class Zscore(Parameter):

def extract(self, sensor):

return Timeseries(

stats.zscore(sensor.cleandata.values),

timestamps=sensor.cleandata.times,

samplerate\_Hz = sensor.cleandata.samplerate\_Hz,

units = "StdDevs"

)

#

###############################################################################

###############################################################################

# Tests

#

class Test():

def \_\_init\_\_(self, trials):

self.scores = dict()

for subject in trials:

self.scores[subject] = self.procedure(trials[subject])

def procedure(self, trial):

return 1

class PrePostCompare(Test):

def \_\_init\_\_(self, trials, events, offset, delay, signal):

self.events = events

self.offset = offset

self.delay = delay

self.signal = signal

self.sampleperiod = 0.01

super().\_\_init\_\_(trials)

def procedure(self, trial):

scores = dict()

for event in self.events:

try:

pre = trial.signals[self.signal].cleandata.interp(self.prerange(trial, event))

post = trial.signals[self.signal].cleandata.interp(self.postrange(trial, event))

scores[event] = self.compare(self.scorepre(pre), self.scorepost(post))

except KeyError:

scores[event] = None

return scores

def prerange(self, trial, event):

trigger = trial.events[event]+self.offset

return np.arange(trigger-self.delay, trigger, self.sampleperiod)

def postrange(self, trial, event):

trigger = trial.events[event]+self.offset

return np.arange(trigger, trigger+self.delay, self.sampleperiod)

def scorepre(self, pre):

return np.sum(pre)

def scorepost(self, post):

return self.scorepre(post)

def compare(self, pre, post):

return (pre-post).astype(float)

class Threshold(PrePostCompare):

def \_\_init\_\_(self, trials, events, offset, delay, signal, threshold):

self.threshold = threshold

super().\_\_init\_\_(trials, events, offset, delay, signal)

def scorepost(self, post):

return np.mean(post)

def compare(self, pre, post):

return np.sign(post)\*(np.abs(post)>self.threshold)

class ThreshPos(Threshold):

def compare(self, pre, post):

return 1\*(post>self.threshold)

class ThreshNeg(Threshold):

def compare(self, pre, post):

return 1\*(post<self.threshold)

class ThreshMax(Threshold):

def scorepost(self, post):

return np.max(post)

class ThreshMin(ThreshMax):

def scorepost(self, post):

return np.max(post)

class BasePostCompare(PrePostCompare):

def prerange(self, trial, event):

baseline\_start = trial.events["BLS"]

if trial.signals[self.signal].cleandata.times[0] > trial.events["BLS"]:

baseline\_start = trial.signals[self.signal].cleandata.times[0]

baseline\_end = trial.events["BLE"]

return np.arange(baseline\_start, baseline\_end, self.sampleperiod)

class PrePostMax(PrePostCompare):

def scorepre(self, pre):

return np.mean(pre)

def scorepost(self, post):

return np.max(post)

class PrePostMin(PrePostCompare):

def scorepre(self, pre):

return np.mean(pre)

def scorepost(self, post):

return np.min(post)

class PrePostRel(PrePostCompare):

def compare(self, pre, post):

return (pre/post).astype(float)-1.0

class BasePostMax(BasePostCompare):

def scorepre(self, pre):

return np.mean(pre)

def scorepost(self, post):

return np.max(post)

class BasePostMin(BasePostCompare):

def scorepre(self, pre):

return np.mean(pre)

def scorepost(self, post):

return np.min(post)

class BasePostRel(BasePostCompare):

def compare(self, pre, post):

return (pre/post).astype(float)-1.0

#

###############################################################################

###############################################################################

# Trial Class

#

class Trial:

def \_\_init\_\_(self, subjectname, rawdata, param):

print("+ Creating "+subjectname)

self.name = subjectname

print(" - Reading event data from file")

rawevents = np.loadtxt(param["eventfile"], dtype=str, delimiter=',', skiprows=2)

print(" - Processing event data")

self.events = dict()

for i in range(len(rawevents)):

self.events[rawevents[i, 1]]=(rawevents[i,0].astype(float))\*60.0 #convert minutes to seconds

times = (rawdata[:,0]\*60)

bp\_data = rawdata[:,1]

rsp\_data = rawdata[:,2]

ppg\_data = rawdata[:,3]

# mbp\_data = rawdata[:,4]

# bpm\_data = rawdata[:,5]

eda\_data = rawdata[:,6]

self.signals = dict()

print(" - Processing Blood Pressure data")

bp = BloodPressure(bp\_data, times, param["bp\_peakdist"], param["bp\_lowbound"])

self.signals[bp.name] = bp

sbp = SystolicPressure("Systolic Pressure", self.signals[bp.name])

self.signals[sbp.name] = sbp

print(" - Processing RSP data")

rsp = RSP(rsp\_data, times)

self.signals[rsp.name] = rsp

rrms = RespRMS("RespRMS", self.signals[rsp.name])

self.signals[rrms.name] = rrms

print(" - Processing PPG data")

ppg = PPG(ppg\_data, times, param["ppg\_peakdist"], param["ppg\_lowbound"])

self.signals[ppg.name] = ppg

ibi = IBI("IBI", self.signals[ppg.name])

self.signals[ibi.name] = ibi

# mbp = AvgBloodPressure(mbp\_data, times)

# self.signals[mbp.name] = mbp

#

# bpm = AvgPulse(bpm\_data, times)

# self.signals[bpm.name] = bpm

print(" - Processing EDA data")

eda = ElectrodermalActivity(eda\_data, times)

self.signals[eda.name] = eda

self.signals["SysPress Z-Score"] = Zscore("SysPress Z-Score", self.signals[sbp.name])

self.signals["RespRMS Z-Score"] = Zscore("RespRMS Z-Score", self.signals[rrms.name])

self.signals["IBI Z-Score"] = Zscore("IBI Z-Score", self.signals[ibi.name])

self.signals["EDA Z-Score"] = Zscore("EDA Z-Score", self.signals[eda.name])

def plot\_events(self, y):

for event in self.events:

plt.annotate(event, (self.events[event], y))

def load\_data(params):

print("+ Loading trial data")

rawdata = dict()

for subject in iter(params):

print(" - loading "+subject)

rawdata[subject] = np.genfromtxt(params[subject]["datafile"], delimiter=' ', skip\_header=17)

return rawdata

def generate\_subjects(params, rawdata):

subjects = dict()

for subject in iter(params):

subjects[subject] = Trial(subject, rawdata[subject], params[subject])

return subjects

#

###############################################################################

###############################################################################

# Experiment Parameters

#

params = { "Subject A": {"datafile": "data/Subject A.txt",

"eventfile": "events/SubjectA\_events.csv",

"bp\_peakdist": 500,

"bp\_lowbound": 50,

"ppg\_peakdist": 500,

"ppg\_lowbound": 0},

"Subject B": {"datafile": "data/Subject B.txt",

"eventfile": "events/SubjectB\_events.csv",

"bp\_peakdist": 300,

"bp\_lowbound": 50,

"ppg\_peakdist": 300,

"ppg\_lowbound": 0},

"Subject C": {"datafile": "data/Subject C.txt",

"eventfile": "events/SubjectC\_events.csv",

"bp\_peakdist": 500,

"bp\_lowbound": 80,

"ppg\_peakdist": 500,

"ppg\_lowbound": 0},

"Subject D": {"datafile": "data/Subject D.txt",

"eventfile": "events/SubjectD\_events.csv",

"bp\_peakdist": 500,

"bp\_lowbound": 100,

"ppg\_peakdist": 500,

"ppg\_lowbound": 0},

"Subject E": {"datafile": "data/Subject E.txt",

"eventfile": "events/SubjectE\_events.csv",

"bp\_peakdist": 500,

"bp\_lowbound": 20,

"ppg\_peakdist": 500,

"ppg\_lowbound": 0},

"Subject F": {"datafile": "data/Subject F.txt",

"eventfile": "events/SubjectF\_events.csv",

"bp\_peakdist": 500,

"bp\_lowbound": 85,

"ppg\_peakdist": 500,

"ppg\_lowbound": 0},

"Subject G": {"datafile": "data/Subject G.txt",

"eventfile": "events/SubjectG\_events.csv",

"bp\_peakdist": 500,

"bp\_lowbound": 65,

"ppg\_peakdist": 500,

"ppg\_lowbound": 0}

}

eventsets = { "Mailbox": ["4B","5B","6B","7B","8B","9B","1C","2C"],

"Bag": ["8C","9C","1D","2D","3D","4D","5D","6D"],

"Cash": ["4B","5B","6B","7B","8C","9C","1D","2D"],

"Check": ["8B","9B","1C","2C","3D","4D","5D","6D"],

"$41": ["6B", "1D", "1C", "5D"],

"$470.16": ["7B","2D","2C","6D"]

}

events\_pertinent = set()

for eventset in eventsets:

events\_pertinent.update(eventsets[eventset])

#rawdata = load\_data(params)

subjects = generate\_subjects(params, rawdata)

quesdelay = 3

eda\_delay = 1

eda\_period = 8

syspress\_delay = 3

syspress\_period = 11

resprms\_delay = 1

resprms\_period = 20

ibi\_delay = 4

ibi\_period = 10

eda\_zthresh = 0.4

syspress\_zthresh = -0.5

resprms\_zthresh = -0.45

ibi\_zthresh = -0.1

tests = {

# "Pre-Post EDA": PrePostRel(subjects, events\_pertinent, quesdelay+eda\_delay, eda\_period, "Electrodermal Activity"),

# "Pre-Post Systolic Pressure": PrePostRel(subjects, events\_pertinent, quesdelay+syspress\_delay, syspress\_period, "Systolic Pressure"),

# "Pre-Post Resp RMS": PrePostRel(subjects, events\_pertinent, quesdelay+resprms\_delay, resprms\_period, "RespRMS"),

# "Pre-Post IBI": PrePostRel(subjects, events\_pertinent, quesdelay+ibi\_delay, ibi\_period, "IBI"),

#

# "Baseline-Post EDA": BasePostRel(subjects, events\_pertinent, quesdelay+eda\_delay, eda\_period, "Electrodermal Activity"),

# "Baseline-Post Systolic Pressure": BasePostRel(subjects, events\_pertinent, quesdelay+syspress\_delay, syspress\_period, "Systolic Pressure"),

# "Baseline-Post Resp RMS": BasePostRel(subjects, events\_pertinent, quesdelay+resprms\_delay, resprms\_period, "RespRMS"),

# "Baseline-Post IBI": BasePostRel(subjects, events\_pertinent, quesdelay+ibi\_delay, ibi\_period, "IBI"),

#

#

# "Pre-Post Mean-Max EDA": PrePostRel(subjects, events\_pertinent, quesdelay+eda\_delay, eda\_period, "Electrodermal Activity"),

# "Pre-Post Mean-Max Systolic Pressure": PrePostRel(subjects, events\_pertinent, quesdelay+syspress\_delay, syspress\_period, "Systolic Pressure"),

# "Pre-Post Mean-Min Resp RMS": PrePostRel(subjects, events\_pertinent, quesdelay+resprms\_delay, resprms\_period, "RespRMS"),

# "Pre-Post Mean-Min IBI": PrePostRel(subjects, events\_pertinent, quesdelay+ibi\_delay, ibi\_period, "IBI"),

#

# "Baseline-Post Mean-Max EDA": BasePostRel(subjects, events\_pertinent, quesdelay+eda\_delay, eda\_period, "Electrodermal Activity"),

# "Baseline-Post Mean-Max Systolic Pressure": BasePostRel(subjects, events\_pertinent, quesdelay+syspress\_delay, syspress\_period, "Systolic Pressure"),

# "Baseline-Post Mean-Min Resp RMS": BasePostRel(subjects, events\_pertinent, quesdelay+resprms\_delay, resprms\_period, "RespRMS"),

# "Baseline-Post Mean-Min IBI": BasePostRel(subjects, events\_pertinent, quesdelay+ibi\_delay, ibi\_period, "IBI"),

#

"Threshold Pos EDA Z-Score": ThreshPos(subjects, events\_pertinent, quesdelay+eda\_delay, eda\_period, "EDA Z-Score", eda\_zthresh),

"Threshold Pos SysPress Z-Score": ThreshPos(subjects, events\_pertinent, quesdelay+syspress\_delay, syspress\_period, "SysPress Z-Score", syspress\_zthresh),

"Threshold Neg RespRms Z-Score": ThreshNeg(subjects, events\_pertinent, quesdelay+resprms\_delay, resprms\_period, "RespRMS Z-Score", resprms\_zthresh),

"Threshold Neg IBI Z-Score": ThreshNeg(subjects, events\_pertinent, quesdelay+ibi\_delay, ibi\_period, "IBI Z-Score", ibi\_zthresh),

}

test\_weights = { "Pre-Post EDA": 1.0,

"Pre-Post Systolic Pressure": 1.0,

"Pre-Post Resp RMS": -1.0,

"Pre-Post IBI": 1.0,

"Baseline-Post EDA": 1.0,

"Baseline-Post Systolic Pressure": 1.0,

"Baseline-Post Resp RMS": -1.0,

"Baseline-Post IBI": 1.0,

"Pre-Post Mean-Max EDA": 1.0,

"Pre-Post Mean-Max Systolic Pressure": 1.0,

"Pre-Post Mean-Min Resp RMS": 1.0,

"Pre-Post Mean-Min IBI": 1.0,

"Baseline-Post Mean-Max EDA": 1.0,

"Baseline-Post Mean-Max Systolic Pressure":1.0,

"Baseline-Post Mean-Min Resp RMS": 1.0,

"Baseline-Post Mean-Min IBI": 1.0,

"Threshold Pos EDA Z-Score": 0.5,

"Threshold Pos SysPress Z-Score": 0.75,

"Threshold Neg RespRms Z-Score": 0.25,

"Threshold Neg IBI Z-Score": 1.0,

}

#

###############################################################################

###############################################################################

# test results

#

test\_scores = dict()

for test in tests:

print("")

print("From "+test+" test results:")

test\_scores[test] = dict()

for subject in subjects:

test\_scores[test][subject] = dict()

for eventset in eventsets:

score = 0.0

for event in eventsets[eventset]:

if tests[test].scores[subject][event] is not None:

score += (tests[test].scores[subject][event])\*test\_weights[test]

test\_scores[test][subject][eventset] = score

place = "Bag"

if test\_scores[test][subject]["Mailbox"] > test\_scores[test][subject]["Bag"]:

place = "Mailbox"

item = "a Check for $470.16"

if (test\_scores[test][subject]["Cash"]+test\_scores[test][subject]["$41"]

> test\_scores[test][subject]["Check"]+test\_scores[test][subject]["$470.16"]):

item = "$41 in Cash"

print(" "+subject+" stole "+item+" from "+place)

print("")

print("Aggregate Test Significance:")

for test in test\_weights:

print(test, test\_weights[test])

print("")

print("Aggregating test results:")

aggregate = dict()

for subject in subjects:

aggregate[subject] = dict()

for eventset in eventsets:

score = 0

for test in tests:

score += test\_scores[test][subject][eventset]

aggregate[subject][eventset] = score

place = "Bag"

if aggregate[subject]["Mailbox"] > aggregate[subject]["Bag"]:

place = "Mailbox"

item = "a Check for $470.16"

if (aggregate[subject]["Cash"]+aggregate[subject]["$41"]

> aggregate[subject]["Check"]+aggregate[subject]["$470.16"]):

item = "$41 Cash"

print(" "+subject+" stole "+item+" from "+place)

#

###############################################################################

###############################################################################

# Plots

#

for subject in subjects.values():

plt.figure(subject.name+" Blood Pressure")

plt.clf()

subject.signals["Blood Pressure"].plot()

subject.signals["Systolic Pressure"].cleandata.plot()

subject.plot\_events(70)

plt.figure(subject.name+" EDA")

plt.clf()

subject.signals["Electrodermal Activity"].plot()

subject.plot\_events(10)

#

plt.figure(subject.name+" RSP")

plt.clf()

subject.signals["RSP"].cleandata.plot()

plt.plot(subject.signals["RSP"].peaks.times, subject.signals["RSP"].peaks.values/np.sqrt(2))

subject.signals["RespRMS"].rawdata.plot()

subject.plot\_events(2)

#

plt.figure(subject.name+" PPG")

plt.clf()

subject.signals["PPG"].plot()

subject.plot\_events(10)

#

plt.figure(subject.name+" IBI")

plt.clf()

subject.signals["IBI"].rawdata.plot()

subject.signals["IBI"].cleandata.plot()

subject.plot\_events(0)

#

# plt.figure(subject.name+"IBI Z-Score")

# subject.signals["IBI Z-Score"].cleandata.plot()

# subject.plot\_events(0)

#

###############################################################################

###############################################################################

# .csv output

#

file = open('./results.csv','w', newline='')

wr = csv.writer(file)

wr.writerow(["== Direct Test Results =="])

for test in tests:

wr.writerow([])

wr.writerow([test])

wr.writerow([" "]+list(tests[test].scores["Subject A"].keys()))

for subject in subjects:

wr.writerow([subject]+list(tests[test].scores[subject].values()))

wr.writerow([])

wr.writerow([])

wr.writerow(["== Question Summed Test Results =="])

for test in test\_scores:

wr.writerow([])

wr.writerow([test])

wr.writerow([" "]+list(test\_scores[test]["Subject A"].keys()))

for subject in subjects:

wr.writerow([subject]+list(test\_scores[test][subject].values()))

wr.writerow([])

wr.writerow([])

wr.writerow(["== All Test Aggregate Results =="])

wr.writerow([])

wr.writerow([test])

wr.writerow([" "]+list(aggregate["Subject A"].keys()))

for subject in subjects:

wr.writerow([subject]+list(aggregate[subject].values()))

file.flush()

file.close()

#

###############################################################################