Evan Magee
Professor Jang
CmpSc 463 - 001
10/29/2025

Overview

This project aims to analyze and cluster time series segments in the PulseDB database. Particularly for the project, it is using 1250 time series segments from 1 subject. The goal is to select one of the three types of signals (ECG/PPG/ABP) and use that to perform divide-and-conquer clustering on said time series segments. Those clusters will then be analyzed to find the closest pair of segments within them and analyze those segments to find the maximum change in pulse over the shortest period of time. Throughout, the project execution textual and visual reports are shown, with a goal of summarizing different aspects of what is being analyzed and to justify decisions made by the algorithms such as clustering and decisions on the closest pair.

How to Use

This project is run solely on through the PulseClusterAnalysis.py file. It requires the PulseDB test subset which will have its file path placed in the path variable at the top of the class. This file is provided in the github. Python packages it requires are numpy, matplotlib, h5py, and dtaidistance. Everything is run in the main method of the file underneath the with h5py.File(path, 'r') as f: statement. Running this file will execute the input functions including clustering, finding closest pairs, and feature extraction using kadane's algorithm. Visualization functions are also included to display the results of these functions.

Structure of Code

The implementation of this program was done entirely in the PulseClusterAnalysis.py which contains clearly defined sections which handle clustering, finding similar points, and extracting features from the PulseDB database. The structure of the code are as follows:

1. Helper Functions
    - This section contains functions that help with correlation, normalization, and segment reduction
2. Divide-and-Conquer Clustering
    - This section contains recursive clustering and splitting functions that split segments into clusters based on their correlation.
3. Closest Pair within Clusters
    - This section contains functions that find the most similar pairs of segments within each cluster based on their dynamic time warping (DTW) distance.
4. Kadane's Algorithm

    - This section contains functions that identify intervals of maximum cumulative change in each time series as well as finding features from them.
5. Visualization
    - This section contains functions that generate plots to show clusters, closest pairs, and kadane intervals
6. Main Method
    - This section contains the entire execution code, data loading, and console output.

Algorithms

Helper Functions

- correlation(x, y)
    - *O(n)*
    - This function returns the correlation values between two given inputs. It does so using the np.corrcoef(x,y)[0,1]
    - It is used for checking the correlation between two segments in the median_pivot_centroid and split_cluster

- normalize_segment(segment)
    - *O(n)*
    - This function returns a Z-score normalized version of the input segment. This is done by first subtracting the given segment by the mean of the segment then that value is divided by the standard deviations of the segment.
        - This is also known a Z-score normalization
    - It helps to reduce redundancy and maintain data integrity.

- reduce_size(segment, factor)
    - *O(n)*
    - This function returns a reduced segment by the input factor to be used for calculations like DTW distance. In its normal size it takes a very long time to execute just one distance so this is required for faster execution.

Divide & Conquer

- median_pivot_centroid(cluster)
    - *O(m * n)*
    - This function, given a cluster, finds the best central segment in the cluster to be used as the pivot for the split_cluster function.

    - It does this by first generating a centroid, then calculating the correlation between all segments in the cluster to that centroid. The segment with the largest correlation to the centroid is output as the pivot.

- split_cluster(cluster, indices, threshold)
    - *O(m * n)*
    - This function takes an input cluster, and splits it along a calculated pivot (done by median_pivot_centroid) , sorting the different segments into two separate clusters based on their correlation to that pivot and whether that correlation is over a certain threshold.
    - Index is also sorted along with this so that segments can have respective id numbers for displaying

- cluster_segments(segments, indices, threshold=0.8, min_size=5)
    - $O(m^2 * n)$
    - This recursive function takes input segments, their indices, a threshold, and minimum cluster size and applies them to split_cluster to produce clusters. The recursion continues only while clusters can still be split and are larger than the input minimum size. The returned list is a tuple of the clusters and their indices.
    - Threshold is passed to split_cluster as the minimum correlation needed for determining which cluster a segment is placed into.

Closest Pair

- closest_pair(cluster, indices)
    - $O(m^2 * n^2)$
    - This function searches through the input cluster to find the 2 closest segments based on dynamic time warping distance.
    - It first checks that the cluster doesn't consist of just one segment.
        - If so then there can't be a closest pair so a tuple of (None, None) and a distance of -1 is returned to show this.
    - If the cluster has 2 or more segments, those segments are iterated though checking the dtw distance, a measure of similarity between 2 time series, between each. If the measured distance is lower than the lowest recorded distance, those 2 segments are set as the closest pair. Once all segments have been compared, the found pair is output along with their dtw distance, as that is the closest pair in the cluster.
    - Indices are used to keep track of the index of the segments so when they are output, they should index relative to all the data rather than just their cluster.

Apply Maximum Subarray

- kadane(segment)
    - *O(n)*
    - This function searches through an input Z-score normalized segment (they were normalized before this but it is required for this function) to find a contiguous subarray with the largest cumulative positive values.
    - It does this by iterating through the readings in the time segment and adding it to the current sum of these readings. This sum then checks to see if it is larger than the maximum sum so far. If so it becomes the new maximum sum and if not, a variable tracking starting value is set to the next array index and the current sum is reset back to zero. This starting value is set to the start when a new maximum is found and the end is set to the current index at the time. Once the entire time-series has been iterated through, maximum sum, start index, and end index are returned. These represent when the maximum change inside the array occurs from and how much of a change it is.

- extract_kadane_features(segments)
    - *O(m * n)*
    - This function serves to extract the features from a given cluster by iterating through its segments and running them through kadane's algorithm. The output max sum is then added to an array. The start and end are also added to the array, but are calculated into what second they start and end of the 10 second segment time-series.
    - These max areas can be interpreted as areas of stress. Since the examples I used cover blood pressure areas with high positive amounts could indicate stress or fear. It could also indicate a lot of physical activity like exercise.

Visualizations

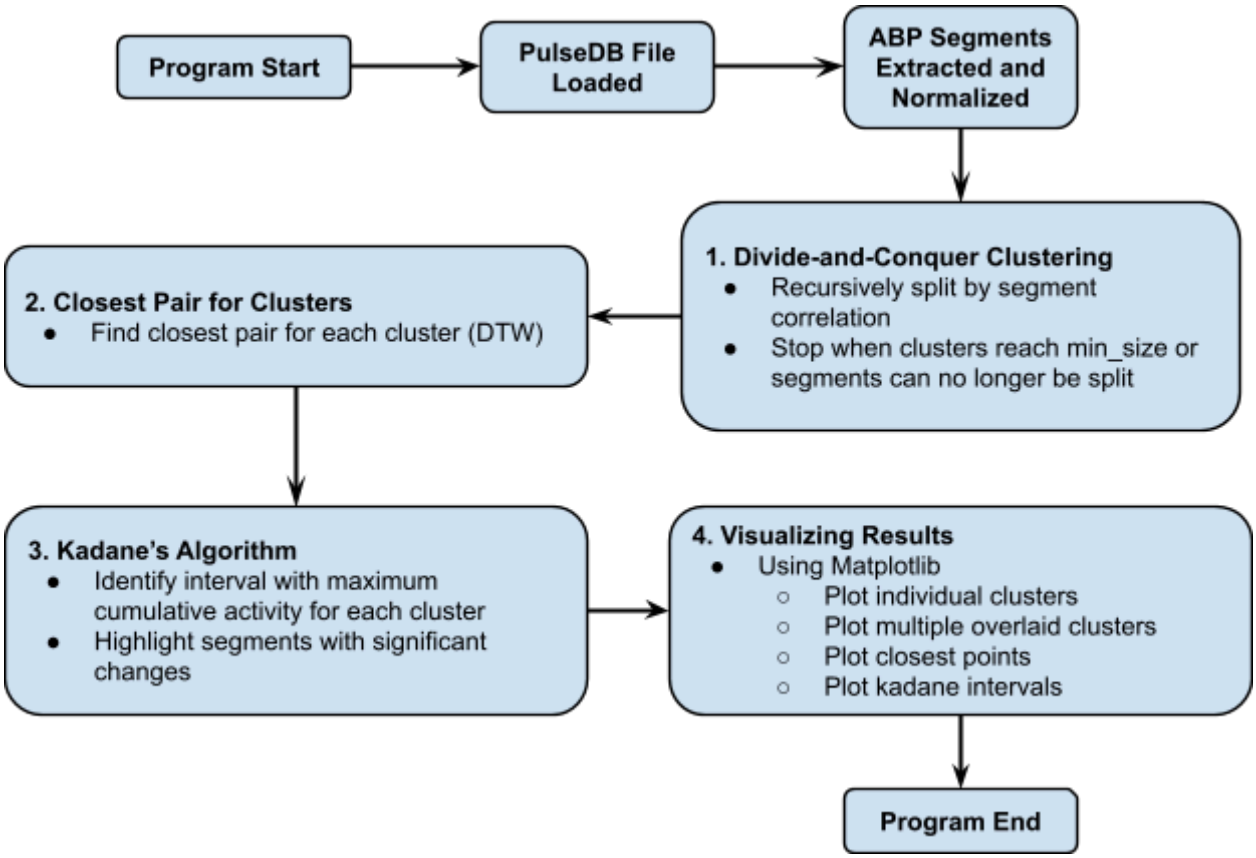
- plot_cluster_examples(clusters, n)
    - *O(m * n)*
    - This function creates a visual plot of n segments for each of the clusters in the input array of clusters. So if you input a list of 3 clusters with n = 7. Then 3 visuals will be made with each containing the first 7 segments of each cluster.
    - This showcases how visually similar each of the segments inside of a given cluster are to show why those segments were clustered together.

- plot_all_clusters(clusters, n, m)
    - *O(m * n)*
    - Similar to plot_cluster_examples but instead of creating separate visual plots, it combines all the first m amount of clusters, and the first n amount of their segments into one plot and overlays the mean of all input segments in purple.
    - This allows for someone to visualize how similar different clusters are on top of each other with different colors allowing for visual differences to appear

- plot_closest_pair(cluster, indices, closest_pair)
    - *O(n)*
    - Given a cluster, its segment indices, and its closest pair, this function visualizes them on top of each other on a time-series plot.
    - This allows you to see the similarities between both segments to see and justify why they are considered closest pairs.

- plot_kadane_interval(segment, start, end)\
    - *O(n)*
    - Given a normalized segment and start & end of the subarray maximum for that segment, this function plots a red highlight overtop of the segment on a time-series plot.
    - This allows you to visually see why that portion of the segment is considered the maximum subarray.

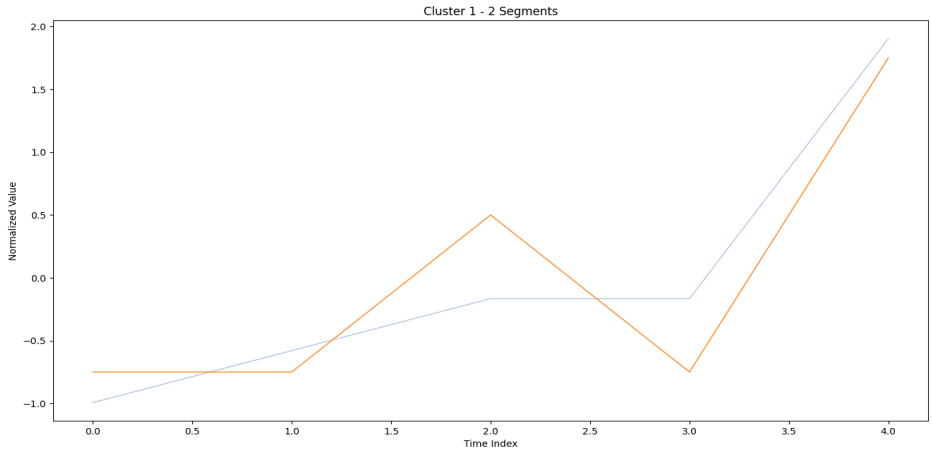
Flowchart

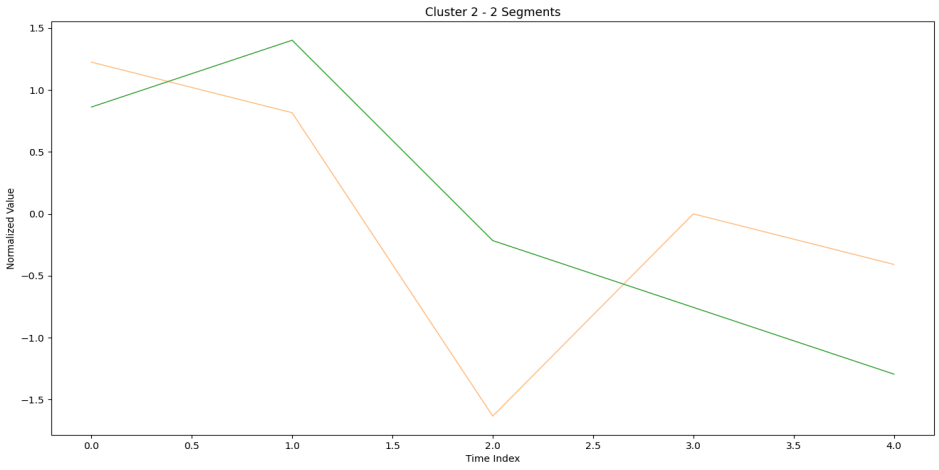

Program Start
PulseDB File Loaded
ABP Segments Extracted and Normalized
1. Divide-and-Conquer Clustering
Recursively split by segment correlation
Stop when clusters reach min_size or segments can no longer be split
2. Closest Pair for Clusters
Find closest pair for each cluster (DTW)
3. Kadane's Algorithm
Identify interval with maximum cumulative activity for each cluster
Highlight segments with significant changes
4. Visualizing Results
Using Matplotlib
Plot individual clusters
Plot multiple overlaid clusters
Plot closest points
Plot kadane intervals
Program End

Toy Example

For the toy example, I created four small time-series style segments, 2 rising and 2 falling, and used them to verify the implemented algorithms. The clustering worked correctly clustering the four segments into 2 separate clusters which have similar results as seen in these graphs:

```
segments= [
    np.array([1, 2, 3, 3, 8]),
    np.array([2, 2, 4 ,2 ,6]),
    np.array([-1, -2, -8, -4, -5]),
    np.array([-2, -1, -4, -5, -6])
]
indices = [0, 1, 2, 3]
```





The closest pair function correctly identified the most similar segments using DTW distance for each cluster. Finally Kadane's algorithm successfully detected the subarray with the largest cumulative positive change. The results from this function confirm that the algorithms are behaving as expected. This can be confirmed further with the output:

```
---- Toy Example Verification ----

---- 1. Divide-and-Conquer Clustering ----
 - Num of Clusters: 2

---- 2. Closest Pair for Clusters ----
 - Cluster 1: Closest pair (0, 1) with DTW distance 0.243
 - Cluster 2: Closest pair (2, 3) with DTW distance 0.362

---- 3. Kadane's Algorithm ----
 - Input: [-2  3  5 -1  4 -5]
 - Max Subarray Sum: 11, Start: 1, End: 4
```

Execution Results

```
---- 1. Divide-and-Conquer Clustering of Time Series Segments ----

    Clustering by ABP
    ------------------------------------------------------
     - Number of ABP Clusters: 81
```

```
---- 2. Closest Pair of Time Series Within Clusters ----

   Computing Closest Pair for Clusters
   -----------------------------------------------
    - Cluster 1: Closest pair (Segment 972, Segment 973) with a distance of 2.629
    - Cluster 2: Closest pair (Segment 559, Segment 560) with a distance of 2.585
    - Cluster 3: Closest pair (Segment 502, Segment 503) with a distance of 2.574
    - Cluster 4: Closest pair (Segment 628, Segment 629) with a distance of 2.660
    - Cluster 5: Closest pair (Segment 316, Segment 317) with a distance of 2.617
    - Cluster 6: Closest pair (Segment 245, Segment 246) with a distance of 2.590
    - Cluster 7: Closest pair (Segment 687, Segment 688) with a distance of 2.595
    - Cluster 8: Closest pair (Segment 215, Segment 216) with a distance of 2.658
    - Cluster 9: Closest pair (Segment 376, Segment 377) with a distance of 2.508
    - Cluster 10: Closest pair (Segment 451, Segment 452) with a distance of 2.603
    - Cluster 11: Closest pair (Segment 407, Segment 408) with a distance of 2.528
    - Cluster 12: Closest pair (Segment 474, Segment 475) with a distance of 2.369
    - Cluster 13: Closest pair (Segment 544, Segment 545) with a distance of 2.623
    - Cluster 14: Closest pair (Segment 20, Segment 21) with a distance of 2.690
    - Cluster 15: Closest pair (Segment 172, Segment 173) with a distance of 2.619
    - Cluster 16: Closest pair (Segment 237, Segment 238) with a distance of 2.453
    - Cluster 17: Closest pair (Segment 272, Segment 273) with a distance of 2.688
    - Cluster 18: Closest pair (Segment 587, Segment 588) with a distance of 2.469
    - Cluster 19: Closest pair (Segment 663, Segment 664) with a distance of 2.560
    - Cluster 20: Closest pair (Segment 626, Segment 627) with a distance of 2.631
```

```
- Cluster 21: Closest pair (Segment 581, Segment 582) with a distance of 2.639
- Cluster 22: Closest pair (Segment 298, Segment 299) with a distance of 2.360
- Cluster 23: Closest pair (Segment 359, Segment 360) with a distance of 2.568
- Cluster 24: Closest pair (Segment 513, Segment 514) with a distance of 2.790
- Cluster 25: Closest pair (Segment 551, Segment 552) with a distance of 2.612
- Cluster 26: Closest pair (Segment 490, Segment 491) with a distance of 2.472
- Cluster 27: Closest pair (Segment 941, Segment 942) with a distance of 2.803
- Cluster 28: Closest pair (Segment 883, Segment 884) with a distance of 2.572
- Cluster 29: Closest pair (Segment 915, Segment 916) with a distance of 2.589
- Cluster 30: Closest pair (Segment 834, Segment 835) with a distance of 2.537
- Cluster 31: Closest pair (Segment 890, Segment 891) with a distance of 2.623
- Cluster 32: Closest pair (Segment 960, Segment 961) with a distance of 2.676
- Cluster 33: Closest pair (Segment 721, Segment 722) with a distance of 2.394
- Cluster 34: Closest pair (Segment 675, Segment 676) with a distance of 2.635
- Cluster 35: Closest pair (Segment 749, Segment 750) with a distance of 2.696
- Cluster 36: Closest pair (Segment 778, Segment 779) with a distance of 2.462
- Cluster 37: Closest pair (Segment 807, Segment 808) with a distance of 2.673
- Cluster 38: Closest pair (Segment 861, Segment 862) with a distance of 2.413
- Cluster 39: Closest pair (Segment 932, Segment 933) with a distance of 2.483
- Cluster 40: Closest pair (Segment 995, Segment 996) with a distance of 2.401
- Cluster 41: Closest pair (Segment 13, Segment 14) with a distance of 2.490
- Cluster 42: Closest pair (Segment 59, Segment 60) with a distance of 2.723
- Cluster 43: Closest pair (Segment 86, Segment 87) with a distance of 2.419
- Cluster 44: Closest pair (Segment 148, Segment 149) with a distance of 2.498
- Cluster 45: Closest pair (Segment 195, Segment 196) with a distance of 2.541
- Cluster 46: Closest pair (Segment 124, Segment 125) with a distance of 2.458
- Cluster 47: Closest pair (Segment 166, Segment 167) with a distance of 2.550
- Cluster 48: Closest pair (Segment 75, Segment 76) with a distance of 2.578
- Cluster 49: Closest pair (Segment 106, Segment 107) with a distance of 2.604
- Cluster 50: Closest pair (Segment 41, Segment 42) with a distance of 2.581
- Cluster 51: Closest pair (Segment 910, Segment 911) with a distance of 2.665
- Cluster 52: Closest pair (Segment 129, Segment 130) with a distance of 2.823
- Cluster 53: Closest pair (Segment 458, Segment 459) with a distance of 2.703
- Cluster 54: Closest pair (Segment 606, Segment 607) with a distance of 2.573
- Cluster 55: Closest pair (Segment 429, Segment 430) with a distance of 2.645
- Cluster 56: Closest pair (Segment 404, Segment 405) with a distance of 2.521
- Cluster 57: Closest pair (Segment 373, Segment 374) with a distance of 2.548
- Cluster 58: Closest pair (Segment 337, Segment 338) with a distance of 2.650
- Cluster 59: Closest pair (Segment 286, Segment 287) with a distance of 2.533
- Cluster 60: Closest pair (Segment 312, Segment 313) with a distance of 2.395
```

```
- Cluster 61: Closest pair (Segment 645, Segment 646) with a distance of 2.680
- Cluster 62: Closest pair (Segment 832, Segment 833) with a distance of 2.624
- Cluster 63: Closest pair (Segment 786, Segment 787) with a distance of 2.576
- Cluster 64: Closest pair (Segment 762, Segment 763) with a distance of 2.565
- Cluster 65: Closest pair (Segment 724, Segment 725) with a distance of 2.567
- Cluster 66: Closest pair (Segment 684, Segment 685) with a distance of 2.620
- Cluster 67: Closest pair (Segment 239, Segment 240) with a distance of 2.548
- Cluster 68: Closest pair (Segment 188, Segment 189) with a distance of 2.540
- Cluster 69: Closest pair (Segment 853, Segment 854) with a distance of 2.733
- Cluster 70: Closest pair (Segment 705, Segment 706) with a distance of 2.818
- Cluster 71: Closest pair (Segment 263, Segment 264) with a distance of 2.745
- Cluster 72: Closest pair (Segment 220, Segment 221) with a distance of 2.993
- Cluster 73: Closest pair (Segment 62, Segment 63) with a distance of 2.763
- Cluster 74: Closest pair (Segment 18, Segment 19) with a distance of 2.778
- Cluster 75: Closest pair (Segment 956, Segment 957) with a distance of 2.790
- Cluster 76: Closest pair (Segment 803, Segment 804) with a distance of 2.861
- Cluster 77: Closest pair (Segment 767, Segment 768) with a distance of 2.663
- Cluster 78: Closest pair (Segment None, Segment None) with a distance of -1.000
- Cluster 79: Closest pair (Segment None, Segment None) with a distance of -1.000
- Cluster 80: Closest pair (Segment None, Segment None) with a distance of -1.000
- Cluster 81: Closest pair (Segment 154, Segment 486) with a distance of 20.613
```

```
---- 3. Apply Maximum Subarray (Kadane's Algorithm) to Each Time Series ----

    Kadane's Algorithm
    --------------------------------------------------

     - Kadane Features Shape: (1000, 3)

    Cluster Reasoning
    --------------------------------------------------
     - Cluster 1: avg_max_sum=826.887, avg_start=8.13s, avg_end=9.55s
     - Cluster 2: avg_max_sum=837.693, avg_start=8.15s, avg_end=9.55s
     - Cluster 3: avg_max_sum=663.206, avg_start=7.80s, avg_end=9.55s
     - Cluster 4: avg_max_sum=691.506, avg_start=5.14s, avg_end=9.54s
     - Cluster 5: avg_max_sum=804.839, avg_start=8.14s, avg_end=9.55s
     - Cluster 6: avg_max_sum=722.458, avg_start=4.00s, avg_end=9.55s
     - Cluster 7: avg_max_sum=735.523, avg_start=2.73s, avg_end=9.52s
     - Cluster 8: avg_max_sum=824.573, avg_start=7.40s, avg_end=9.54s
     - Cluster 9: avg_max_sum=636.616, avg_start=4.64s, avg_end=9.55s
     - Cluster 10: avg_max_sum=843.244, avg_start=7.51s, avg_end=9.55s
```

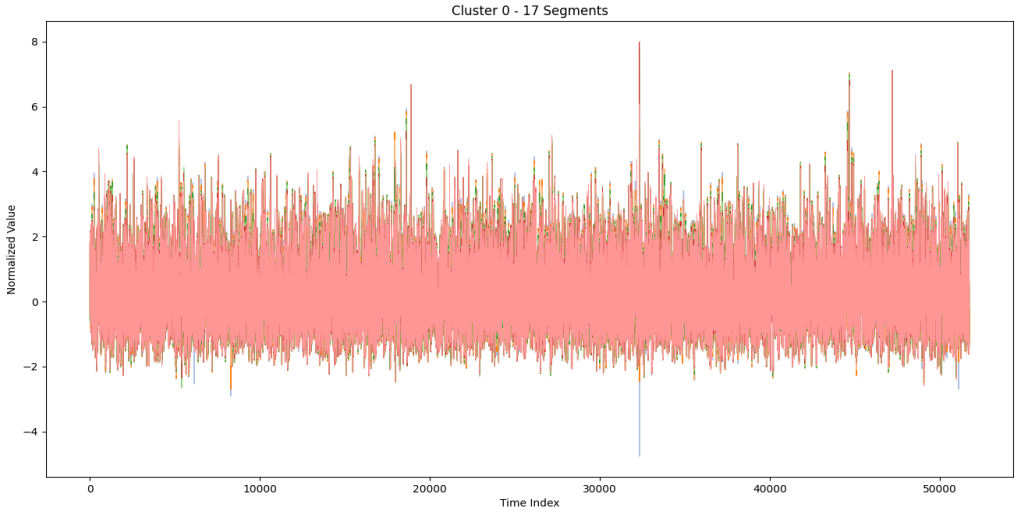
```
- Cluster 11: avg_max_sum=694.277, avg_start=8.14s, avg_end=9.54s
- Cluster 12: avg_max_sum=817.391, avg_start=8.12s, avg_end=9.55s
- Cluster 13: avg_max_sum=654.131, avg_start=7.83s, avg_end=9.54s
- Cluster 14: avg_max_sum=748.649, avg_start=5.49s, avg_end=9.55s
- Cluster 15: avg_max_sum=657.876, avg_start=7.48s, avg_end=9.55s
- Cluster 16: avg_max_sum=820.199, avg_start=8.14s, avg_end=9.55s
- Cluster 17: avg_max_sum=637.899, avg_start=6.09s, avg_end=9.55s
- Cluster 18: avg_max_sum=838.881, avg_start=8.12s, avg_end=9.55s
- Cluster 19: avg_max_sum=722.102, avg_start=7.83s, avg_end=9.55s
- Cluster 20: avg_max_sum=722.370, avg_start=8.12s, avg_end=9.53s
- Cluster 21: avg_max_sum=884.735, avg_start=8.15s, avg_end=9.55s
- Cluster 22: avg_max_sum=816.457, avg_start=7.83s, avg_end=9.54s
- Cluster 23: avg_max_sum=795.565, avg_start=6.87s, avg_end=9.55s
- Cluster 24: avg_max_sum=632.610, avg_start=7.76s, avg_end=9.54s
- Cluster 25: avg_max_sum=715.433, avg_start=2.73s, avg_end=9.53s
- Cluster 26: avg_max_sum=723.102, avg_start=8.11s, avg_end=9.55s
- Cluster 27: avg_max_sum=719.436, avg_start=6.06s, avg_end=9.55s
- Cluster 28: avg_max_sum=800.780, avg_start=8.12s, avg_end=9.55s
- Cluster 29: avg_max_sum=629.441, avg_start=8.20s, avg_end=9.53s
- Cluster 30: avg_max_sum=795.319, avg_start=8.14s, avg_end=9.55s
- Cluster 31: avg_max_sum=711.615, avg_start=8.19s, avg_end=9.55s
- Cluster 32: avg_max_sum=765.747, avg_start=8.14s, avg_end=9.55s
- Cluster 33: avg_max_sum=765.367, avg_start=7.84s, avg_end=9.53s
- Cluster 34: avg_max_sum=718.926, avg_start=7.52s, avg_end=9.55s
- Cluster 35: avg_max_sum=806.575, avg_start=8.14s, avg_end=9.55s
- Cluster 36: avg_max_sum=715.616, avg_start=8.15s, avg_end=9.55s
- Cluster 37: avg_max_sum=654.288, avg_start=8.15s, avg_end=9.53s
- Cluster 38: avg_max_sum=815.859, avg_start=8.14s, avg_end=9.55s
- Cluster 39: avg_max_sum=675.037, avg_start=8.15s, avg_end=9.55s
- Cluster 40: avg_max_sum=884.222, avg_start=8.12s, avg_end=9.55s
- Cluster 41: avg_max_sum=786.020, avg_start=3.09s, avg_end=9.54s
- Cluster 42: avg_max_sum=693.490, avg_start=2.84s, avg_end=9.55s
- Cluster 43: avg_max_sum=833.670, avg_start=8.13s, avg_end=9.54s
- Cluster 44: avg_max_sum=723.868, avg_start=6.23s, avg_end=9.55s
- Cluster 45: avg_max_sum=779.962, avg_start=5.09s, avg_end=9.55s
```

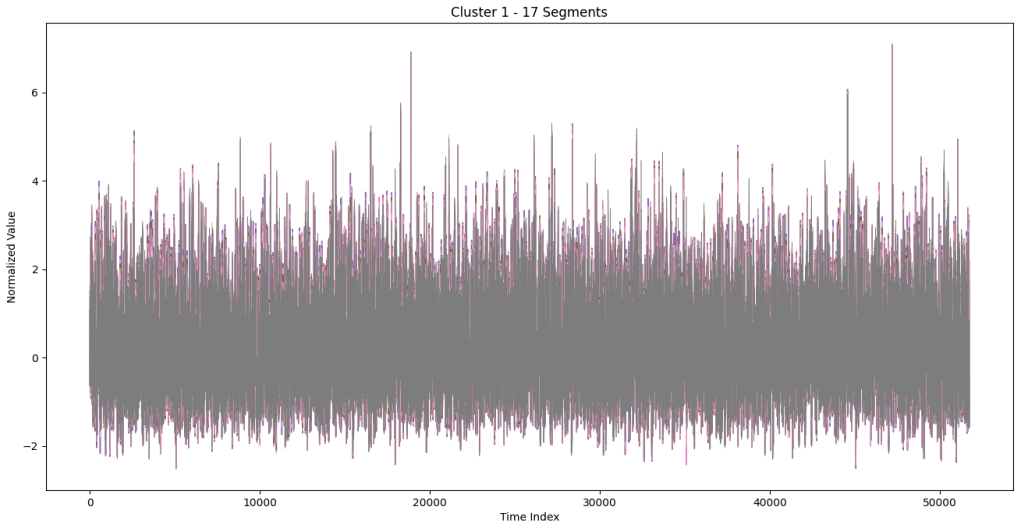
```
- Cluster 46: avg_max_sum=824.246, avg_start=6.07s, avg_end=9.55s
- Cluster 47: avg_max_sum=599.979, avg_start=8.14s, avg_end=9.55s
- Cluster 48: avg_max_sum=636.164, avg_start=7.49s, avg_end=9.53s
- Cluster 49: avg_max_sum=825.854, avg_start=8.14s, avg_end=9.55s
- Cluster 50: avg_max_sum=688.885, avg_start=2.77s, avg_end=9.55s
- Cluster 51: avg_max_sum=619.750, avg_start=8.60s, avg_end=9.55s
- Cluster 52: avg_max_sum=861.715, avg_start=2.56s, avg_end=9.55s
- Cluster 53: avg_max_sum=802.550, avg_start=4.39s, avg_end=9.55s
- Cluster 54: avg_max_sum=732.572, avg_start=8.11s, avg_end=9.55s
- Cluster 55: avg_max_sum=826.848, avg_start=8.14s, avg_end=9.55s
- Cluster 56: avg_max_sum=647.913, avg_start=8.13s, avg_end=9.55s
- Cluster 57: avg_max_sum=743.159, avg_start=2.73s, avg_end=9.55s
- Cluster 58: avg_max_sum=835.708, avg_start=8.15s, avg_end=9.55s
- Cluster 59: avg_max_sum=717.846, avg_start=8.14s, avg_end=9.53s
- Cluster 60: avg_max_sum=825.786, avg_start=7.23s, avg_end=9.55s
- Cluster 61: avg_max_sum=701.150, avg_start=2.73s, avg_end=9.55s
- Cluster 62: avg_max_sum=714.862, avg_start=8.14s, avg_end=9.54s
- Cluster 63: avg_max_sum=662.658, avg_start=5.60s, avg_end=9.55s
- Cluster 64: avg_max_sum=725.516, avg_start=8.12s, avg_end=9.55s
- Cluster 65: avg_max_sum=800.978, avg_start=5.62s, avg_end=9.55s
- Cluster 66: avg_max_sum=703.693, avg_start=2.73s, avg_end=9.53s
- Cluster 67: avg_max_sum=756.344, avg_start=8.14s, avg_end=9.55s
- Cluster 68: avg_max_sum=737.124, avg_start=8.14s, avg_end=9.55s
- Cluster 69: avg_max_sum=838.240, avg_start=8.14s, avg_end=9.55s
- Cluster 70: avg_max_sum=723.305, avg_start=4.53s, avg_end=9.47s
- Cluster 71: avg_max_sum=705.864, avg_start=2.73s, avg_end=9.55s
- Cluster 72: avg_max_sum=814.757, avg_start=8.13s, avg_end=9.53s
- Cluster 73: avg_max_sum=605.541, avg_start=2.73s, avg_end=9.55s
- Cluster 74: avg_max_sum=748.830, avg_start=2.27s, avg_end=9.55s
- Cluster 75: avg_max_sum=727.590, avg_start=3.09s, avg_end=9.55s
- Cluster 76: avg_max_sum=602.811, avg_start=8.15s, avg_end=9.55s
- Cluster 77: avg_max_sum=651.549, avg_start=8.14s, avg_end=9.53s
- Cluster 78: avg_max_sum=745.960, avg_start=8.11s, avg_end=9.52s
- Cluster 79: avg_max_sum=914.458, avg_start=8.15s, avg_end=9.55s
- Cluster 80: avg_max_sum=834.200, avg_start=8.14s, avg_end=9.55s
- Cluster 81: avg_max_sum=722.686, avg_start=7.05s, avg_end=9.55s
```

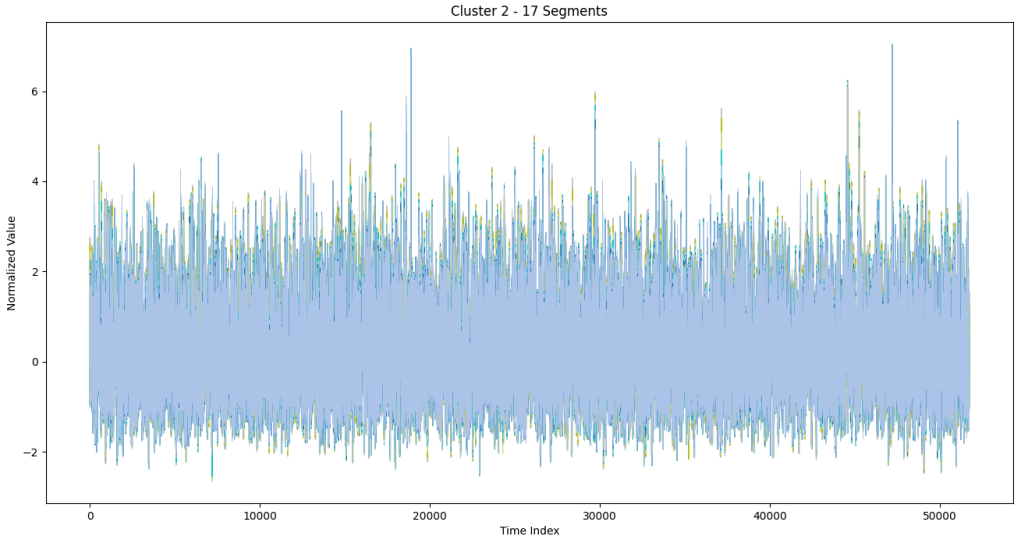
```
Highlighted Events
--------------------------------------------------
Found  50 Significant Events:
 - Segment 91: max_change=868.888, start=8.12s, end=9.55s
 - Segment 92: max_change=877.986, start=8.12s, end=9.55s
 - Segment 93: max_change=880.616, start=8.12s, end=9.55s
 - Segment 94: max_change=880.720, start=8.12s, end=9.55s
 - Segment 95: max_change=875.365, start=8.12s, end=9.55s
 - Segment 96: max_change=866.025, start=8.14s, end=9.55s
 - Segment 127: max_change=865.357, start=2.27s, end=9.55s
 - Segment 128: max_change=880.763, start=2.27s, end=9.55s
 - Segment 129: max_change=889.472, start=2.27s, end=9.55s
 - Segment 130: max_change=888.239, start=2.27s, end=9.55s
 - Segment 131: max_change=881.926, start=2.27s, end=9.55s
 - Segment 132: max_change=871.661, start=2.73s, end=9.55s
 - Segment 434: max_change=866.546, start=8.14s, end=9.55s
 - Segment 435: max_change=868.182, start=8.14s, end=9.55s
 - Segment 436: max_change=869.839, start=8.14s, end=9.55s
 - Segment 437: max_change=870.871, start=8.14s, end=9.55s
 - Segment 438: max_change=872.527, start=8.14s, end=9.55s
 - Segment 439: max_change=871.328, start=8.14s, end=9.55s
 - Segment 440: max_change=867.761, start=8.15s, end=9.55s
 - Segment 441: max_change=867.817, start=8.15s, end=9.55s
 - Segment 442: max_change=866.128, start=8.15s, end=9.55s
 - Segment 563: max_change=865.215, start=8.15s, end=9.55s
 - Segment 564: max_change=867.268, start=8.15s, end=9.55s
 - Segment 577: max_change=877.448, start=8.15s, end=9.55s
 - Segment 578: max_change=888.914, start=8.15s, end=9.55s
 - Segment 579: max_change=899.448, start=8.15s, end=9.55s
 - Segment 580: max_change=908.711, start=8.15s, end=9.55s
 - Segment 581: max_change=914.909, start=8.15s, end=9.55s
 - Segment 582: max_change=916.646, start=8.15s, end=9.55s
 - Segment 583: max_change=914.458, start=8.15s, end=9.55s
 - Segment 584: max_change=910.469, start=8.15s, end=9.55s
 - Segment 585: max_change=904.373, start=8.12s, end=9.55s
 - Segment 586: max_change=896.887, start=8.12s, end=9.55s
 - Segment 587: max_change=888.913, start=8.12s, end=9.55s
 - Segment 588: max_change=879.679, start=8.12s, end=9.55s
 - Segment 589: max_change=873.467, start=8.12s, end=9.55s
 - Segment 590: max_change=868.647, start=8.12s, end=9.55s
 - Segment 984: max_change=871.219, start=8.12s, end=9.55s
```

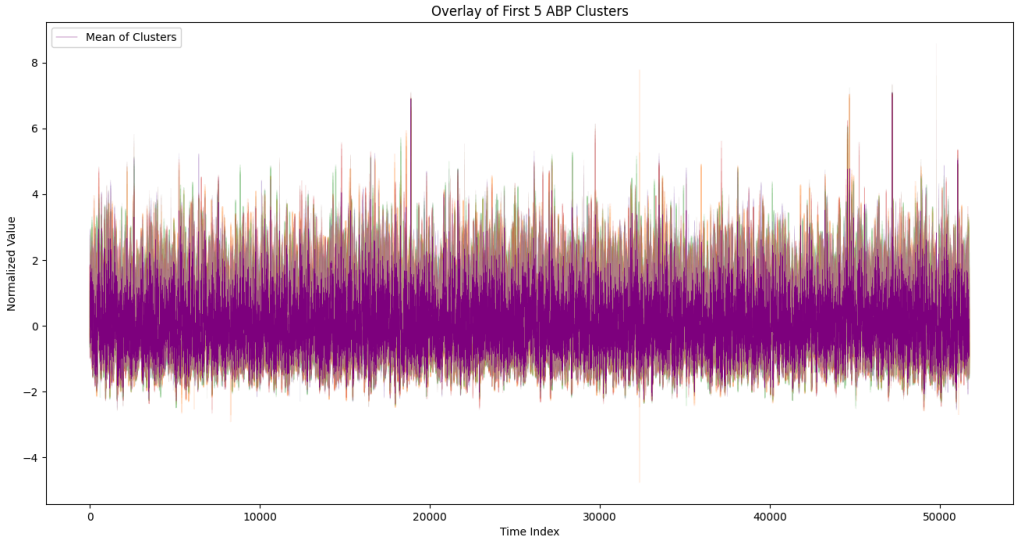
```
    - Segment 985: max_change=884.033, start=8.12s, end=9.55s
    - Segment 986: max_change=894.186, start=8.12s, end=9.55s
    - Segment 987: max_change=901.188, start=8.12s, end=9.55s
    - Segment 988: max_change=904.977, start=8.12s, end=9.55s
    - Segment 989: max_change=907.003, start=8.12s, end=9.55s
    - Segment 990: max_change=907.245, start=8.12s, end=9.55s
    - Segment 991: max_change=905.282, start=8.12s, end=9.55s
    - Segment 992: max_change=904.816, start=8.12s, end=9.55s
    - Segment 993: max_change=902.195, start=8.12s, end=9.55s
    - Segment 994: max_change=894.609, start=8.12s, end=9.55s
    - Segment 995: max_change=888.268, start=8.12s, end=9.55s
    - Segment 996: max_change=879.360, start=8.12s, end=9.55s

---- Step 4: Generate Reports and Visualizations ----

   Visualizing First 3 Clusters Individually
   --------------------------------------------------

   Visualizing First 10 Clusters Together
   --------------------------------------------------
```

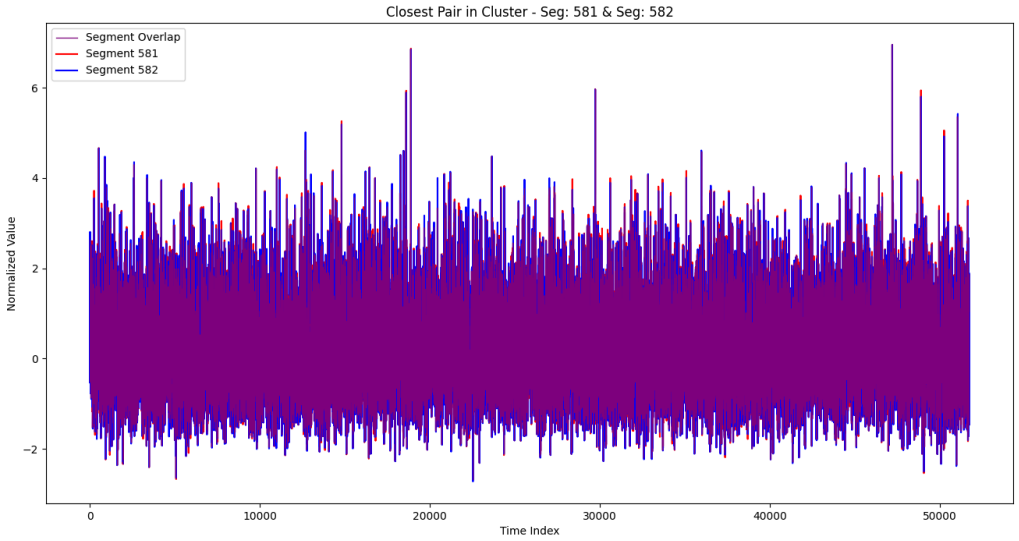
```
  Visualizing 3 Sets of  Closest Pair
  --------------------------------------------------

  Visualizing Maximum Subarray Intervals for 3 Segments
  --------------------------------------------------
```

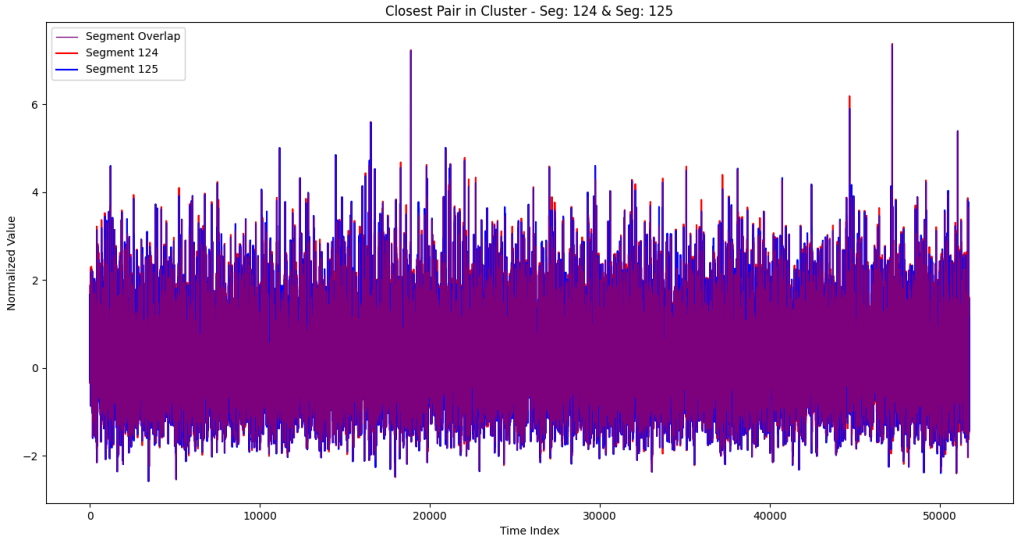


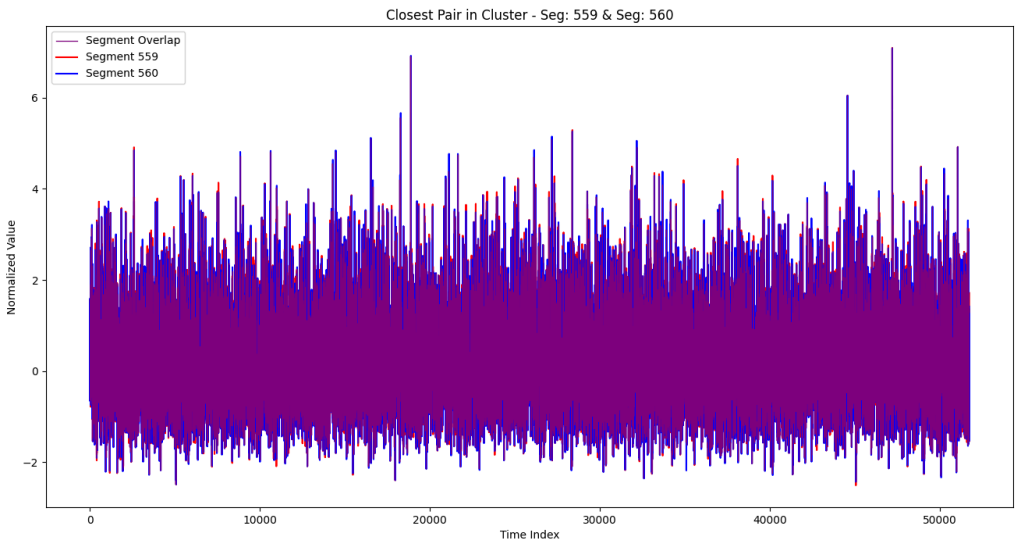

Cluster 1 - 17 Segments
Normalized Value
Time Index
6
4
2
0
−2
0
10000
20000
30000
40000
50000

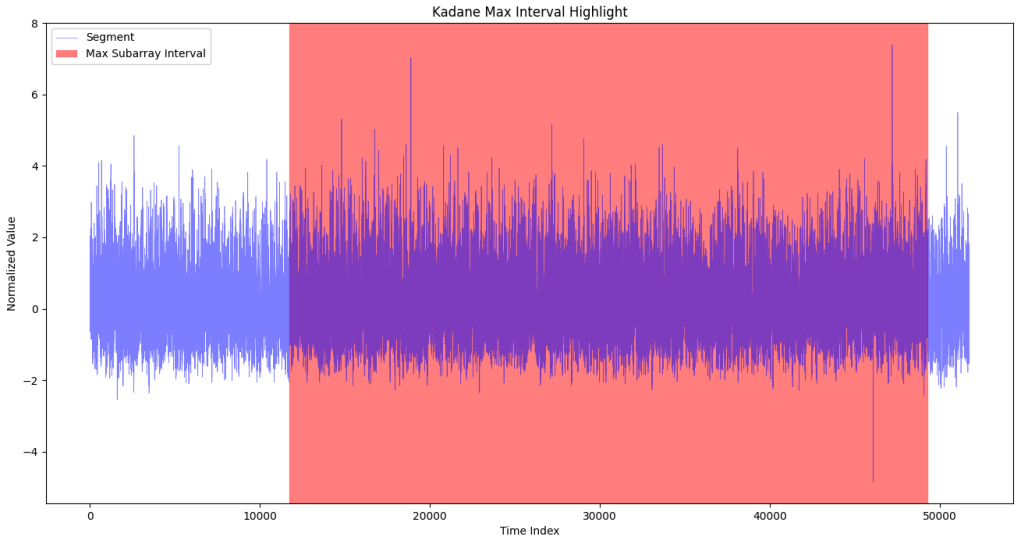

Cluster 2 - 17 Segments
Normalized Value
Time Index
6
4
2
0
−2
0
10000
20000
30000
40000
50000

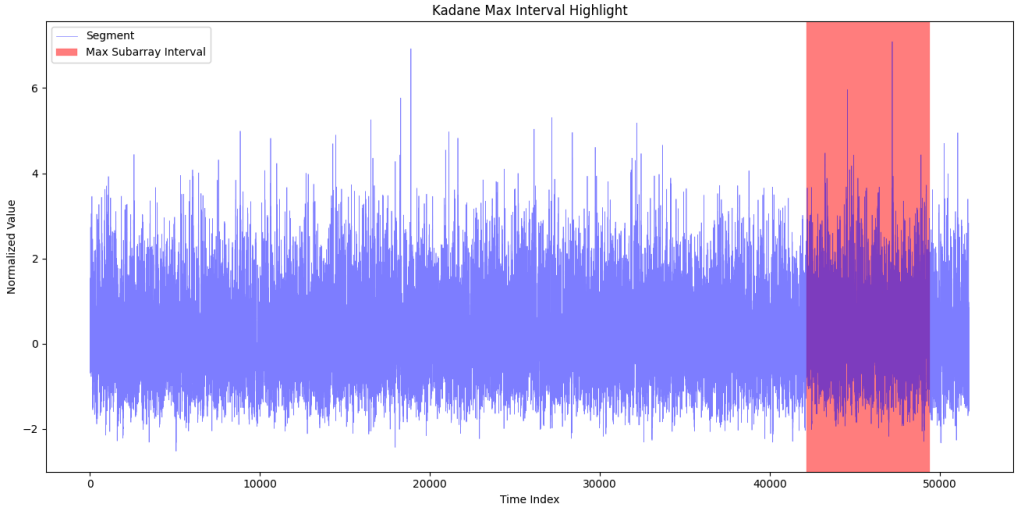

Overlay of First 5 ABP Clusters
Mean of Clusters
Normalized Value
Time Index
8
6
4
2
0
−2
−4
0
10000
20000
30000
40000
50000

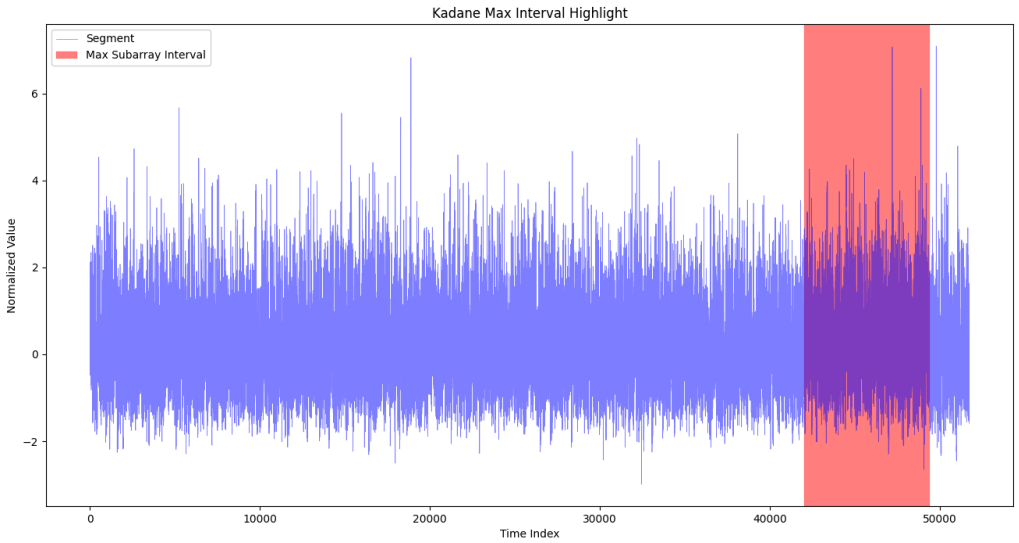

Closest Pair in Cluster - Seg: 581 & Seg: 582
Segment Overlap
Segment 581
Segment 582
Normalized Value
Time Index


Closest Pair in Cluster - Seg: 124 & Seg: 125
Segment Overlap
Segment 124
Segment 125
Normalized Value
Time Index


Closest Pair in Cluster - Seg: 559 & Seg: 560
Segment Overlap
Segment 559
Segment 560
Normalized Value
Time Index


Kadane Max Interval Highlight
Segment
Max Subarray Interval
Normalized Value
Time Index
8
6
4
2
0
−2
−4
0
10000
20000
30000
40000
50000


Kadane Max Interval Highlight
Segment
Max Subarray Interval
Normalized Value
Time Index
6
4
2
0
−2
0
10000
20000
30000
40000
50000


Kadane Max Interval Highlight
Segment
Max Subarray Interval
Normalized Value
Time Index
6
4
2
0
−2
0
10000
20000
30000
40000
50000

Execution Discussion

- **Clustering**
    - The first piece of data from the execution I thought was cool to see was that the clustering turned 1000 segments into 81 clusters showing there are a lot of similar pieces of data within the readings. It also showed that a threshold of 0.8 and minimum cluster size of 5 provided well balanced and pure clusters. Viewing the cluster plots, it provides clear reasoning to why those time-series were grouped together.
- **Closest Pair**
    - From results of the closest pairs functions, I found there were a lot of interesting patterns present in the data output. One thing I noticed was that most of the closest pairs were neighboring points as if they were one segment index off of each other. This was present with all closest pairs but I feel that may be with how close the data was taken together rather than an error in the reading. However, points 78 to 80 showed no pairs being found or having none. This means that those points are not a part of a cluster and may be outliers. Point 81 also showed a distance of 20 which is larger than any of the other DTW distances indicating it is most likely an outlying cluster. Other pairs all had similar distance which indicates good pairing.
- **Kadane's Algorithm**
    - From the results for the Kadane's algorithm, I noticed that a lot of the regions showed high activity from around 8.12s to 9.55s, which may indicate a natural cycle in blood pressure every 10 seconds. However, instances like in the last interval highlight plot seem to showcase times of maybe stress or a period of higher physical activity than normal or potential stress. The period ending around the same time of 9.55s could also be something instrument related where it becomes inaccurate towards the end of the 10 second segment.
- **Highlighted Events**
    - Kadane's algorithm also helped to identify significant events during the time-series which were featured within the 95th percentile of all of the algorithm's results. This could be areas that had longer physical activity or stress as they occurred within specific ranges of the segments. These ranges include segments 91-96, 127-132, 434-442, 563-564, 577-590, and 984-996. These are areas that showcase the largest increases in blood pressure in the subject which can indicate a stress response or even abnormal heart rhythms also known as arrhythmia.

Challenges

There were plenty of challenges faced in this project especially in the understanding and implementation of PulseDB. Its structure was sometimes confusing and a lot of things didn't fully click until they were visualized, especially in what the data was actually representing. While this project only used 1000 entries, this amount could be hundreds if not thousands of times larger so there are definitely areas that could be improved to deal with much larger datasets. One of the largest challenges faced was dealing with the code's runtime. For actions like calculating DTW distance, It at times took minutes to calculate for the first 2 segments due to the large amount of readings present in each. It was almost abandoned in favor of just keeping correlation through reduction and parameter tweaking, a much faster execution was found.

The clustering process also presented challenges towards the beginning, often grouping segments with bias towards the first segments. It was later realized that this was the result of the pivot chosen which was just to use the first segment of the input cluster. This resulted in unbalanced and non-coherent clusters which were undesirable. Through different pivot methods and testing, it was found that the best results were creating 81 clusters instead of the 103 clusters from using the first segment. While these challenges weren't extremely difficult, they definitely helped to provide lessons on improving the performance of different algorithms and code.

Conclusions

This project successfully showcased how using divide-and-conquer strategies and algorithmic reasoning can be used to analyze and interpret biomedical time-series data. Using divide-and-conquer it allowed for similar ABP time-series to be placed into coherent clusters to allow for easier analysis of similar waveforms for tasks such as finding the closest pair without the need for relying on ML libraries. These pairs along with visualizations helped to validate these clusters, which could then be used for Kadane's algorithm. This algorithm helped to identify areas of interest in each segment and from these identify regions which contain significant events of interest. These high-activity events can then be used to help identify things like stress or even health issues such as arrhythmia or irregular heartbeats. Overall, this project had some challenges but demonstrated how powerful these relatively simple algorithms can be when used in the real world, especially in biomedical data.