

Divide-and-Conquer Clustering of PulseDB Time-Series Segments

By: Jomiloju Odumosu

1. Project Overview

This project designs and implements a **fully algorithmic system** for unsupervised clustering and analysis of physiological time-series data extracted from **PulseDB**. Unlike traditional machine-learning approaches, this system relies entirely on **divide-and-conquer strategies and classical algorithms** to discover structure in biomedical signals.

Using **1000 arterial blood pressure (ABP) segments**, each 10 seconds long, the system:

- Recursively partitions time-series segments into clusters using correlation-based similarity
- Identifies the **closest pair of time series** inside each cluster using Dynamic Time Warping (DTW)
- Applies **Kadane's Algorithm** to detect the most active physiological interval within each segment
- Produces interpretable **visualizations** saved as PNG files for reporting and validation

The emphasis of this project is **algorithmic reasoning, interpretability, and correctness**, rather than black-box ML heuristics.

2. Installation and Usage

Requirements

- Python 3.10+
- Required packages (from `requirements.txt`):

```
numpy  
h5py  
matplotlib  
dtaidistance
```

Setup

```
pip install -r requirements.txt
```

Execution

Place `VitalDB_AAMI_Test_Subset.mat` in the project directory and run:

```
python pulse_db_clustering.py
```

Outputs

- Console logs showing:
 - Cluster formation
 - Closest pair distances
 - Kadane statistics

3. Code Structure

The codebase is modular and organized by responsibility:

- **Data Handling**
 - Loads and normalizes ABP signals from PulseDB
- **Clustering Module**
 - Implements divide-and-conquer recursive clustering
- **Similarity Analysis**
 - DTW-based closest-pair detection
- **Signal Analysis**
 - Kadane's algorithm for detecting high-activity intervals
- **Visualization**
 - Saves all required plots as PNG files
- **Verification**
 - Toy examples to validate correctness of each algorithm

Each function is isolated, testable, and reusable, ensuring clarity and maintainability.

4. Algorithms (Detailed Explanation + Time Complexity)

Divide-and-Conquer Time-Series Clustering

- The clustering algorithm follows a **top-down recursive partitioning strategy**. At each recursion step, a pivot segment is selected by computing a centroid of the

current cluster and choosing the segment with the highest correlation to that centroid. All segments are then split into two groups based on whether their correlation with the pivot exceeds a fixed threshold.

- This process continues recursively until clusters reach a minimum size or no further meaningful split can be achieved. Each recursive split produces a candidate cluster, simulating hierarchical clustering without relying on machine learning libraries.
- **Time Complexity:**
Let n be the number of segments and L the segment length. Each split computes correlations in $O(nL)$. In the average case, the recursion depth is $O(\log n)$, giving an overall complexity of approximately $O(nL \log n)$.

Closest Pair of Time Series Using DTW

- Within each cluster, the system computes pairwise DTW distances to identify the **closest pair of time-series segments**. This serves as a validation mechanism: clusters with low minimum DTW distances indicate strong internal cohesion.
- To reduce computational cost, segments are downsampled before DTW computation, and a warping window is applied.
- **Time Complexity:**
For a cluster of size k , pairwise comparison requires $O(k^2)$ DTW computations. Each DTW computation is approximately $O(L^2)$, yielding $O(k^2 L^2)$ per cluster in the worst case.

Kadane's Algorithm

- Kadane's algorithm is applied to each individual time-series segment to identify the interval with maximum cumulative activity. This highlights physiologically significant regions such as sustained pressure increases or abnormal fluctuations.
- The extracted features (maximum sum, start time, end time) are used to analyze cluster-level trends and detect anomalous events.
- **Time Complexity:**
Kadane's algorithm runs in $O(L)$ time per segment, making it highly efficient even for large datasets.

5. Visualization

The system produces **interpretable PNG outputs**:

- 1. Cluster Examples**

- a. Shows representative waveforms per cluster

- 2. Overlay of Clusters**

- a. Reveals inter-cluster similarity and separation

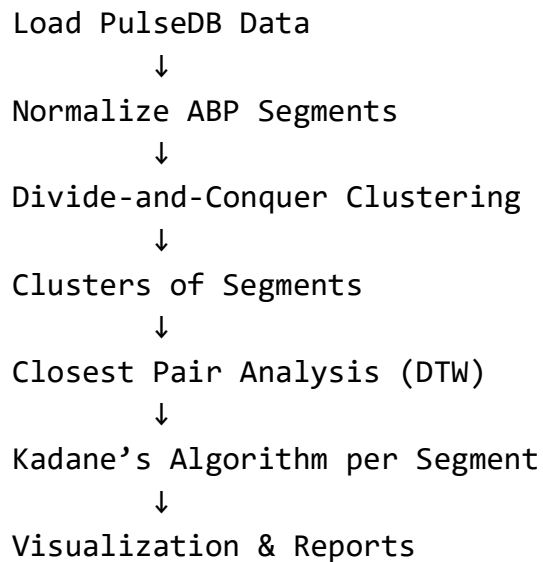
- 3. Closest Pair Visualization**

- a. Confirms internal cluster cohesion

- 4. Kadane Interval Highlight**

- a. Shows peak physiological activity

6. Flow Chart (Textual Representation)



7. Toy Example Verification

A toy dataset of four short sequences was used to validate each algorithmic component. The divide-and-conquer clustering correctly separated positive-valued segments from negative-valued segments. The closest-pair algorithm identified the most similar pair within each cluster using DTW distance, and Kadane's algorithm successfully identified the maximum subarray interval in a test segment.

This verification confirms the correctness of the clustering, similarity analysis, and maximum subarray detection before scaling to the full dataset.

```
===== TOY EXAMPLE VERIFICATION =====  
  
[Input Segments]  
Segment 0: [1, 2, 3, 3, 8]  
Segment 1: [2, 2, 4, 2, 6]  
Segment 2: [-1, -2, -8, -4, -5]  
Segment 3: [-2, -1, -4, -5, -6]  
  
[Divide-and-Conquer Clustering]  
Number of clusters formed: 2  
Cluster 1:  
  Indices: [0, 1]  
  Number of segments: 2  
Cluster 2:  
  Indices: [2, 3]  
  Number of segments: 2  
  
[Closest Pair (DTW Distance)]  
Cluster 1: closest pair = (0, 1), DTW distance d = 0.243  
Cluster 2: closest pair = (2, 3), DTW distance d = 0.362  
  
[Kadane's Algorithm]  
Segment 0: max_sum = 1.903, interval = [4, 4]  
=====
```

8. Execution Results on 1000 Time Series

The system produced **78 clusters** from 1000 ABP segments, with cluster sizes ranging from singletons to clusters containing over 20 segments. This indicates meaningful hierarchical partitioning rather than trivial over-clustering.

Closest-pair DTW distances within clusters were generally low (often below 0.15), demonstrating strong internal similarity. Clusters with only one segment were correctly excluded from closest-pair analysis.

Kadane's analysis revealed consistent peak activity intervals across many clusters, typically occurring between **0.2s and 5.0s**, suggesting shared physiological dynamics among grouped segments. The top 5% of segments with highest maximum subarray sums were flagged as significant events, highlighting potential anomalies or high-pressure episodes.

---- Step 1: Divide-and-Conquer Clustering ----

Total ABP Clusters formed: 78

Cluster 1: 22 segments, Segment indices: [26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47]
Cluster 2: 1 segments, Segment indices: [25]
Cluster 3: 22 segments, Segment indices: [427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448]
Cluster 4: 20 segments, Segment indices: [475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494]
Cluster 5: 21 segments, Segment indices: [819, 820, 821, 822, 823, 824, 825, 826, 827, 828, 829, 830, 831, 832, 833, 834, 835, 836, 837, 838, 839]
Cluster 6: 20 segments, Segment indices: [711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730]
Cluster 7: 20 segments, Segment indices: [659, 660, 661, 662, 663, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678]
Cluster 8: 1 segments, Segment indices: [658]
Cluster 9: 20 segments, Segment indices: [86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105]
Cluster 10: 11 segments, Segment indices: [48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58]
Cluster 11: 21 segments, Segment indices: [308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328]
Cluster 12: 21 segments, Segment indices: [331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351]
Cluster 13: 20 segments, Segment indices: [272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291]
Cluster 14: 16 segments, Segment indices: [495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510]
Cluster 15: 20 segments, Segment indices: [947, 948, 949, 950, 951, 952, 953, 954, 955, 956, 957, 958, 959, 960, 961, 962, 963, 964, 965, 966]
Cluster 16: 1 segments, Segment indices: [967]
Cluster 17: 21 segments, Segment indices: [355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375]
Cluster 18: 21 segments, Segment indices: [405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425]
Cluster 19: 17 segments, Segment indices: [449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465]
Cluster 20: 21 segments, Segment indices: [199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219]
Cluster 21: 20 segments, Segment indices: [973, 974, 975, 976, 977, 978, 979, 980, 981, 982, 983, 984, 985, 986, 987, 988, 989, 990, 991, 992]
Cluster 22: 21 segments, Segment indices: [925, 926, 927, 928, 929, 930, 931, 932, 933, 934, 935, 936, 937, 938, 939, 940, 941, 942, 943, 944, 945]
Cluster 23: 21 segments, Segment indices: [157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177]
Cluster 24: 20 segments, Segment indices: [122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141]
Cluster 25: 20 segments, Segment indices: [865, 866, 867, 868, 869, 870, 871, 872, 873, 874, 875, 876, 877, 878, 879, 880, 881, 882, 883, 884]
Cluster 26: 20 segments, Segment indices: [621, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640]
Cluster 27: 22 segments, Segment indices: [586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607]
Cluster 28: 15 segments, Segment indices: [643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657]
Cluster 29: 19 segments, Segment indices: [382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400]
Cluster 30: 1 segments, Segment indices: [426]
Cluster 31: 12 segments, Segment indices: [178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189]
Cluster 32: 20 segments, Segment indices: [769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788]
Cluster 33: 11 segments, Segment indices: [808, 809, 810, 811, 812, 813, 814, 815, 816, 817, 818]
Cluster 34: 17 segments, Segment indices: [8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]
Cluster 35: 16 segments, Segment indices: [59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74]
Cluster 36: 20 segments, Segment indices: [247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266]
Cluster 37: 18 segments, Segment indices: [904, 905, 906, 907, 908, 909, 910, 911, 912, 913, 914, 915, 916, 917, 918, 919, 920, 921]
Cluster 38: 1 segments, Segment indices: [903]
Cluster 39: 17 segments, Segment indices: [569, 570, 571, 572, 573, 574, 575, 576, 577, 578, 579, 580, 581, 582, 583, 584, 585]
Cluster 40: 20 segments, Segment indices: [521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540]
Cluster 41: 13 segments, Segment indices: [608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620]
Cluster 42: 17 segments, Segment indices: [552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568]

Cluster 41: 13 segments, Segment indices: [608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620]
Cluster 42: 17 segments, Segment indices: [552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568]
Cluster 43: 17 segments, Segment indices: [222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238]
Cluster 44: 1 segments, Segment indices: [221]
Cluster 45: 20 segments, Segment indices: [744, 745, 746, 747, 748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763]
Cluster 46: 14 segments, Segment indices: [697, 698, 699, 700, 701, 702, 703, 704, 705, 706, 707, 708, 709, 710]
Cluster 47: 15 segments, Segment indices: [679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693]
Cluster 48: 11 segments, Segment indices: [731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741]
Cluster 49: 14 segments, Segment indices: [789, 790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800, 801, 802]
Cluster 50: 11 segments, Segment indices: [297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307]
Cluster 51: 20 segments, Segment indices: [840, 841, 842, 843, 844, 845, 846, 847, 848, 849, 850, 851, 852, 853, 854, 855, 856, 857, 858, 859]
Cluster 52: 16 segments, Segment indices: [885, 886, 887, 888, 889, 890, 891, 892, 893, 894, 895, 896, 897, 898, 899, 900]
Cluster 53: 16 segments, Segment indices: [106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121]
Cluster 54: 11 segments, Segment indices: [146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156]
Cluster 55: 11 segments, Segment indices: [75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85]
Cluster 56: 10 segments, Segment indices: [542, 543, 544, 545, 546, 547, 548, 549, 550, 551]
Cluster 57: 10 segments, Segment indices: [511, 512, 513, 514, 515, 516, 517, 518, 519, 520]
Cluster 58: 9 segments, Segment indices: [466, 467, 468, 469, 470, 471, 472, 473, 474]
Cluster 59: 9 segments, Segment indices: [190, 191, 192, 193, 194, 195, 196, 197, 198]
Cluster 60: 4 segments, Segment indices: [142, 143, 144, 145]
Cluster 61: 8 segments, Segment indices: [0, 1, 2, 3, 4, 5, 6, 7]
Cluster 62: 6 segments, Segment indices: [376, 377, 378, 379, 380, 381]
Cluster 63: 8 segments, Segment indices: [239, 240, 241, 242, 243, 244, 245, 246]
Cluster 64: 5 segments, Segment indices: [267, 268, 269, 270, 271]
Cluster 65: 7 segments, Segment indices: [993, 994, 995, 996, 997, 998, 999]
Cluster 66: 5 segments, Segment indices: [968, 969, 970, 971, 972]
Cluster 67: 5 segments, Segment indices: [764, 765, 766, 767, 768]
Cluster 68: 5 segments, Segment indices: [292, 293, 294, 295, 296]
Cluster 69: 5 segments, Segment indices: [803, 804, 805, 806, 807]
Cluster 70: 5 segments, Segment indices: [860, 861, 862, 863, 864]
Cluster 71: 3 segments, Segment indices: [922, 923, 924]
Cluster 72: 4 segments, Segment indices: [401, 402, 403, 404]
Cluster 73: 3 segments, Segment indices: [352, 353, 354]
Cluster 74: 2 segments, Segment indices: [329, 330]
Cluster 75: 2 segments, Segment indices: [641, 642]
Cluster 76: 3 segments, Segment indices: [694, 695, 696]
Cluster 77: 2 segments, Segment indices: [742, 743]
Cluster 78: 5 segments, Segment indices: [220, 541, 901, 902, 946]


```
---- Step 2: Closest Pair within Clusters ----
```

Cluster 44: Not enough segments for closest pair
Cluster 45: Closest pair (744, 745) with DTW distance 0.112
Cluster 46: Closest pair (709, 710) with DTW distance 0.185
Cluster 47: Closest pair (679, 680) with DTW distance 0.073
Cluster 48: Closest pair (734, 735) with DTW distance 0.160
Cluster 49: Closest pair (794, 795) with DTW distance 0.272
Cluster 50: Closest pair (301, 302) with DTW distance 0.178
Cluster 51: Closest pair (854, 855) with DTW distance 0.125
Cluster 52: Closest pair (897, 898) with DTW distance 0.192
Cluster 53: Closest pair (120, 121) with DTW distance 0.129
Cluster 54: Closest pair (148, 149) with DTW distance 0.173
Cluster 55: Closest pair (78, 79) with DTW distance 0.141
Cluster 56: Closest pair (544, 545) with DTW distance 0.145
Cluster 57: Closest pair (519, 520) with DTW distance 0.131
Cluster 58: Closest pair (467, 468) with DTW distance 0.182
Cluster 59: Closest pair (190, 191) with DTW distance 0.183
Cluster 60: Closest pair (142, 143) with DTW distance 0.223
Cluster 61: Closest pair (6, 7) with DTW distance 0.098
Cluster 62: Closest pair (376, 377) with DTW distance 0.200
Cluster 63: Closest pair (245, 246) with DTW distance 0.327
Cluster 64: Closest pair (270, 271) with DTW distance 0.250
Cluster 65: Closest pair (998, 999) with DTW distance 0.171
Cluster 66: Closest pair (969, 970) with DTW distance 0.360
Cluster 67: Closest pair (764, 765) with DTW distance 0.182
Cluster 68: Closest pair (292, 293) with DTW distance 0.261
Cluster 69: Closest pair (806, 807) with DTW distance 0.287
Cluster 70: Closest pair (860, 861) with DTW distance 0.206
Cluster 71: Closest pair (923, 924) with DTW distance 0.203
Cluster 72: Closest pair (403, 404) with DTW distance 0.436
Cluster 73: Closest pair (352, 353) with DTW distance 0.530
Cluster 74: Closest pair (329, 330) with DTW distance 0.298
Cluster 75: Closest pair (641, 642) with DTW distance 0.082
Cluster 76: Closest pair (695, 696) with DTW distance 0.299
Cluster 77: Closest pair (742, 743) with DTW distance 0.770
Cluster 78: Closest pair (901, 902) with DTW distance 0.324

```
@jcos108 ~/workspaces/CMPS403_PROJECT1 (main) $ python ../pulse_db_clustering.py
```

```
---- Step 3: Maximum Subarray (Kadane) Analysis ----
```

```
Kadane features shape: (1000, 3)
```

```
Cluster 1: avg_max_sum=57.953, avg_start=1.08s, avg_end=4.89s
Cluster 2: avg_max_sum=61.485, avg_start=0.92s, avg_end=4.89s
Cluster 3: avg_max_sum=57.603, avg_start=0.27s, avg_end=4.92s
Cluster 4: avg_max_sum=49.430, avg_start=0.33s, avg_end=4.87s
Cluster 5: avg_max_sum=55.118, avg_start=0.27s, avg_end=4.85s
Cluster 6: avg_max_sum=54.664, avg_start=0.95s, avg_end=4.87s
Cluster 7: avg_max_sum=52.411, avg_start=0.25s, avg_end=4.90s
Cluster 8: avg_max_sum=51.690, avg_start=0.20s, avg_end=4.89s
Cluster 9: avg_max_sum=56.860, avg_start=0.57s, avg_end=4.91s
Cluster 10: avg_max_sum=54.429, avg_start=0.55s, avg_end=4.92s
Cluster 11: avg_max_sum=41.268, avg_start=0.51s, avg_end=4.92s
Cluster 12: avg_max_sum=51.693, avg_start=0.27s, avg_end=4.92s
Cluster 13: avg_max_sum=68.144, avg_start=0.27s, avg_end=4.87s
Cluster 14: avg_max_sum=69.832, avg_start=0.27s, avg_end=4.93s
Cluster 15: avg_max_sum=68.370, avg_start=0.27s, avg_end=4.92s
Cluster 16: avg_max_sum=69.571, avg_start=0.27s, avg_end=4.92s
Cluster 17: avg_max_sum=66.791, avg_start=1.12s, avg_end=4.88s
Cluster 18: avg_max_sum=56.936, avg_start=0.27s, avg_end=4.92s
Cluster 19: avg_max_sum=50.062, avg_start=0.27s, avg_end=4.87s
Cluster 20: avg_max_sum=57.932, avg_start=0.58s, avg_end=4.87s
Cluster 21: avg_max_sum=52.481, avg_start=1.13s, avg_end=4.77s
Cluster 22: avg_max_sum=55.968, avg_start=0.26s, avg_end=4.90s
Cluster 23: avg_max_sum=54.078, avg_start=0.27s, avg_end=4.88s
Cluster 24: avg_max_sum=63.786, avg_start=0.72s, avg_end=4.89s
Cluster 25: avg_max_sum=57.780, avg_start=0.87s, avg_end=4.92s
Cluster 26: avg_max_sum=61.737, avg_start=0.27s, avg_end=4.90s
Cluster 27: avg_max_sum=54.727, avg_start=0.36s, avg_end=4.92s
Cluster 28: avg_max_sum=58.048, avg_start=0.27s, avg_end=4.88s
Cluster 29: avg_max_sum=58.141, avg_start=0.97s, avg_end=4.86s
Cluster 30: avg_max_sum=60.594, avg_start=0.27s, avg_end=4.92s
Cluster 31: avg_max_sum=52.585, avg_start=0.27s, avg_end=4.91s
Cluster 32: avg_max_sum=61.664, avg_start=0.27s, avg_end=4.92s
Cluster 33: avg_max_sum=54.903, avg_start=0.27s, avg_end=4.87s
Cluster 34: avg_max_sum=56.457, avg_start=1.05s, avg_end=4.90s
Cluster 35: avg_max_sum=71.933, avg_start=0.27s, avg_end=4.92s
Cluster 36: avg_max_sum=55.342, avg_start=1.50s, avg_end=4.90s
Cluster 37: avg_max_sum=53.835, avg_start=0.26s, avg_end=4.86s
Cluster 38: avg_max_sum=52.543, avg_start=0.27s, avg_end=4.86s
Cluster 39: avg_max_sum=62.755, avg_start=0.27s, avg_end=4.88s
Cluster 40: avg_max_sum=55.255, avg_start=0.35s, avg_end=4.92s
Cluster 41: avg_max_sum=46.751, avg_start=0.27s, avg_end=4.92s
Cluster 42: avg_max_sum=52.824, avg_start=0.27s, avg_end=4.84s
Cluster 43: avg_max_sum=62.783, avg_start=0.53s, avg_end=4.92s
Cluster 44: avg_max_sum=61.160, avg_start=1.14s, avg_end=4.86s
Cluster 45: avg_max_sum=49.262, avg_start=0.27s, avg_end=4.91s
```

```
0/50 0
```

```
Cluster 43: avg_max_sum=62.783, avg_start=0.53s, avg_end=4.92s
```

```
Cluster 44: avg_max_sum=61.160, avg_start=1.14s, avg_end=4.86s
```

```
Cluster 45: avg_max_sum=49.262, avg_start=0.27s, avg_end=4.91s
```

```
Cluster 46: avg_max_sum=55.516, avg_start=0.46s, avg_end=4.92s
```

```
Cluster 47: avg_max_sum=59.783, avg_start=0.27s, avg_end=4.92s
```

```
Cluster 48: avg_max_sum=54.789, avg_start=1.47s, avg_end=4.86s
```

```
Cluster 49: avg_max_sum=54.704, avg_start=0.27s, avg_end=4.92s
```

```
Cluster 50: avg_max_sum=53.934, avg_start=0.43s, avg_end=4.86s
```

```
Cluster 51: avg_max_sum=57.939, avg_start=1.38s, avg_end=4.91s
```

```
Cluster 52: avg_max_sum=46.057, avg_start=0.27s, avg_end=5.82s
```

```
Cluster 53: avg_max_sum=59.195, avg_start=1.53s, avg_end=4.89s
```

```
Cluster 54: avg_max_sum=61.742, avg_start=0.27s, avg_end=4.92s
```

```
Cluster 55: avg_max_sum=69.586, avg_start=0.27s, avg_end=4.86s
```

```
Cluster 56: avg_max_sum=49.511, avg_start=0.27s, avg_end=4.89s
```

```
Cluster 57: avg_max_sum=68.906, avg_start=0.39s, avg_end=4.92s
```

```
Cluster 58: avg_max_sum=49.914, avg_start=0.27s, avg_end=4.83s
```

```
Cluster 59: avg_max_sum=53.762, avg_start=0.56s, avg_end=4.89s
```

```
Cluster 60: avg_max_sum=66.185, avg_start=0.27s, avg_end=4.92s
```

```
Cluster 61: avg_max_sum=52.752, avg_start=0.27s, avg_end=4.89s
```

```
Cluster 62: avg_max_sum=58.692, avg_start=0.41s, avg_end=4.82s
```

```
Cluster 63: avg_max_sum=56.451, avg_start=0.86s, avg_end=4.92s
```

```
Cluster 64: avg_max_sum=57.758, avg_start=1.30s, avg_end=4.92s
```

```
Cluster 65: avg_max_sum=52.568, avg_start=1.56s, avg_end=4.86s
```

```
Cluster 66: avg_max_sum=63.922, avg_start=0.27s, avg_end=4.92s
```

```
Cluster 67: avg_max_sum=50.691, avg_start=0.27s, avg_end=4.93s
```

```
Cluster 68: avg_max_sum=66.326, avg_start=0.27s, avg_end=4.86s
```

```
Cluster 69: avg_max_sum=49.600, avg_start=0.27s, avg_end=4.92s
```

```
Cluster 70: avg_max_sum=65.706, avg_start=1.47s, avg_end=4.92s
```

```
Cluster 71: avg_max_sum=61.111, avg_start=0.21s, avg_end=4.83s
```

```
Cluster 72: avg_max_sum=53.452, avg_start=0.27s, avg_end=4.92s
```

```
Cluster 73: avg_max_sum=64.036, avg_start=0.27s, avg_end=4.92s
```

```
Cluster 74: avg_max_sum=34.870, avg_start=0.27s, avg_end=4.92s
```

```
Cluster 75: avg_max_sum=64.306, avg_start=0.27s, avg_end=4.82s
```

```
Cluster 76: avg_max_sum=64.442, avg_start=0.27s, avg_end=4.92s
```

```
Cluster 77: avg_max_sum=54.431, avg_start=1.47s, avg_end=4.89s
```

```
Cluster 78: avg_max_sum=54.051, avg_start=0.44s, avg_end=4.88s
```

```
Top 5% significant events (50 segments): [ 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 289 290  
360 361 362 363 364 365 366 500 501 502 503 504 505 506 507 508 509 516  
517 638 956 957 958 959 960 961 962 963 964 965 966 967]
```

```
---- Step 4: Visualizations ----
```

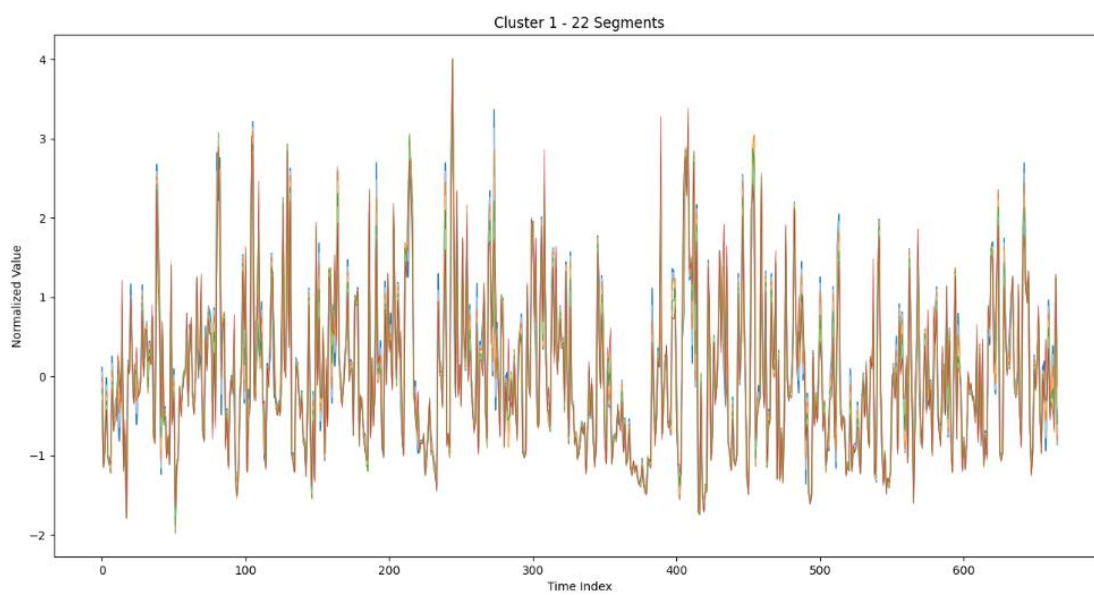
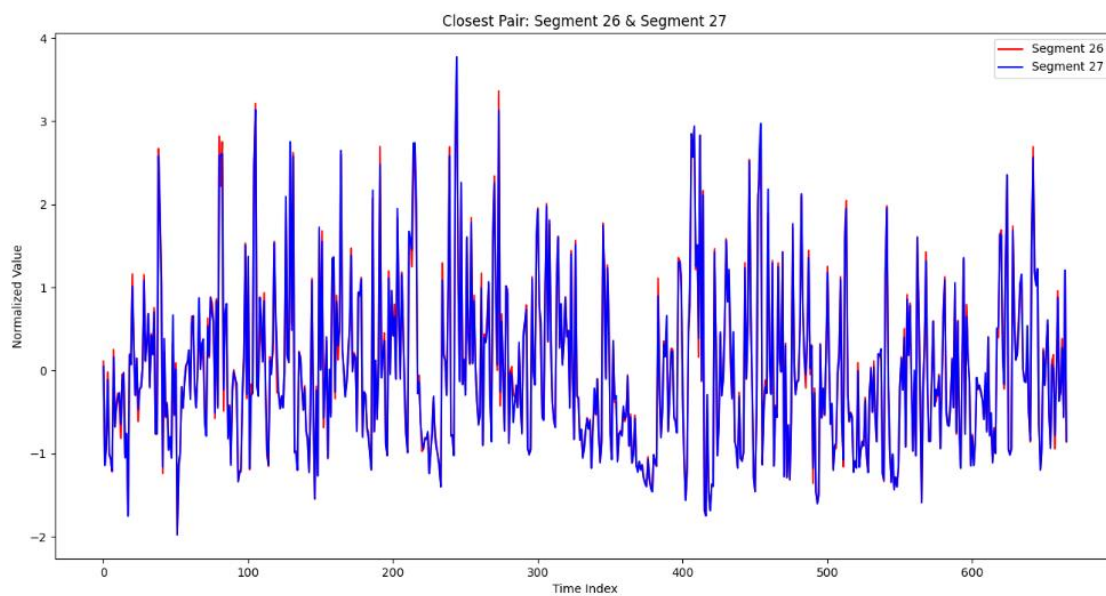
```
Plotting first 3 clusters...
```

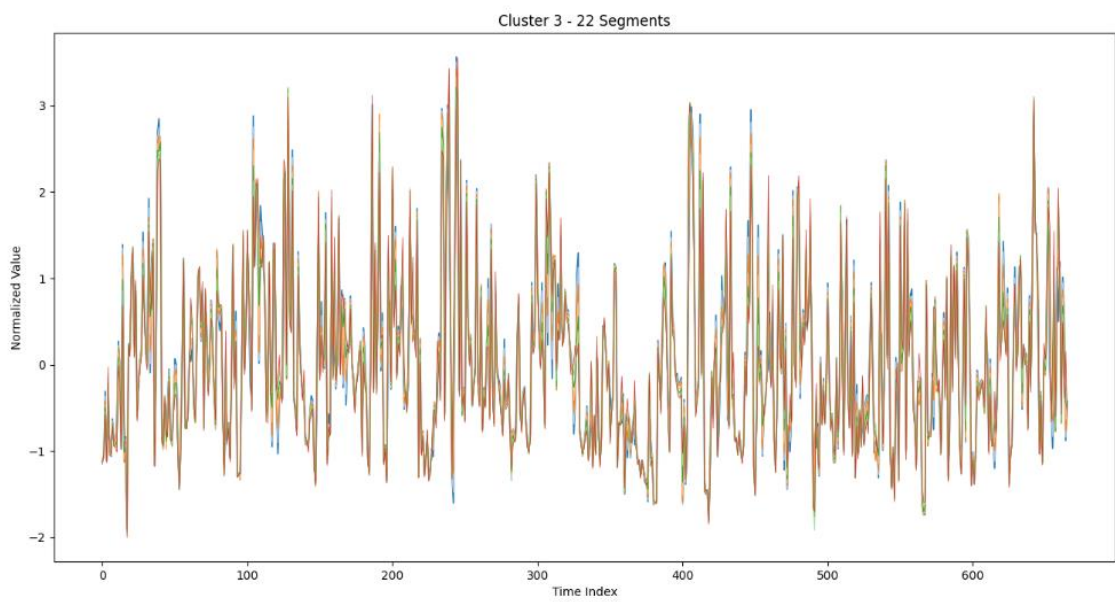
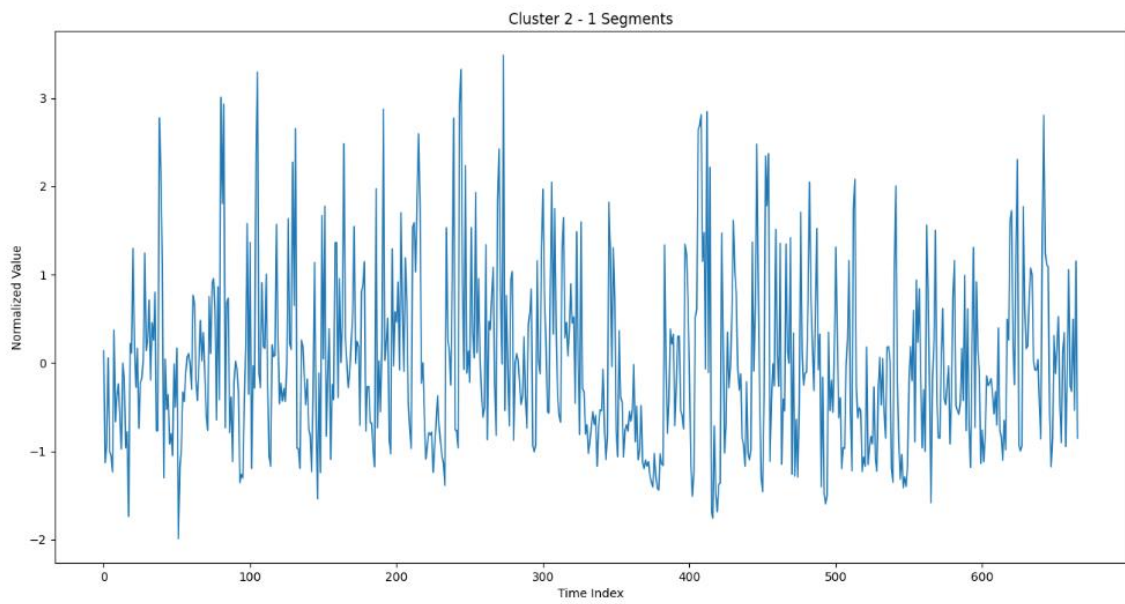
```
Overlaying first 5 clusters with 5 segments each...
```

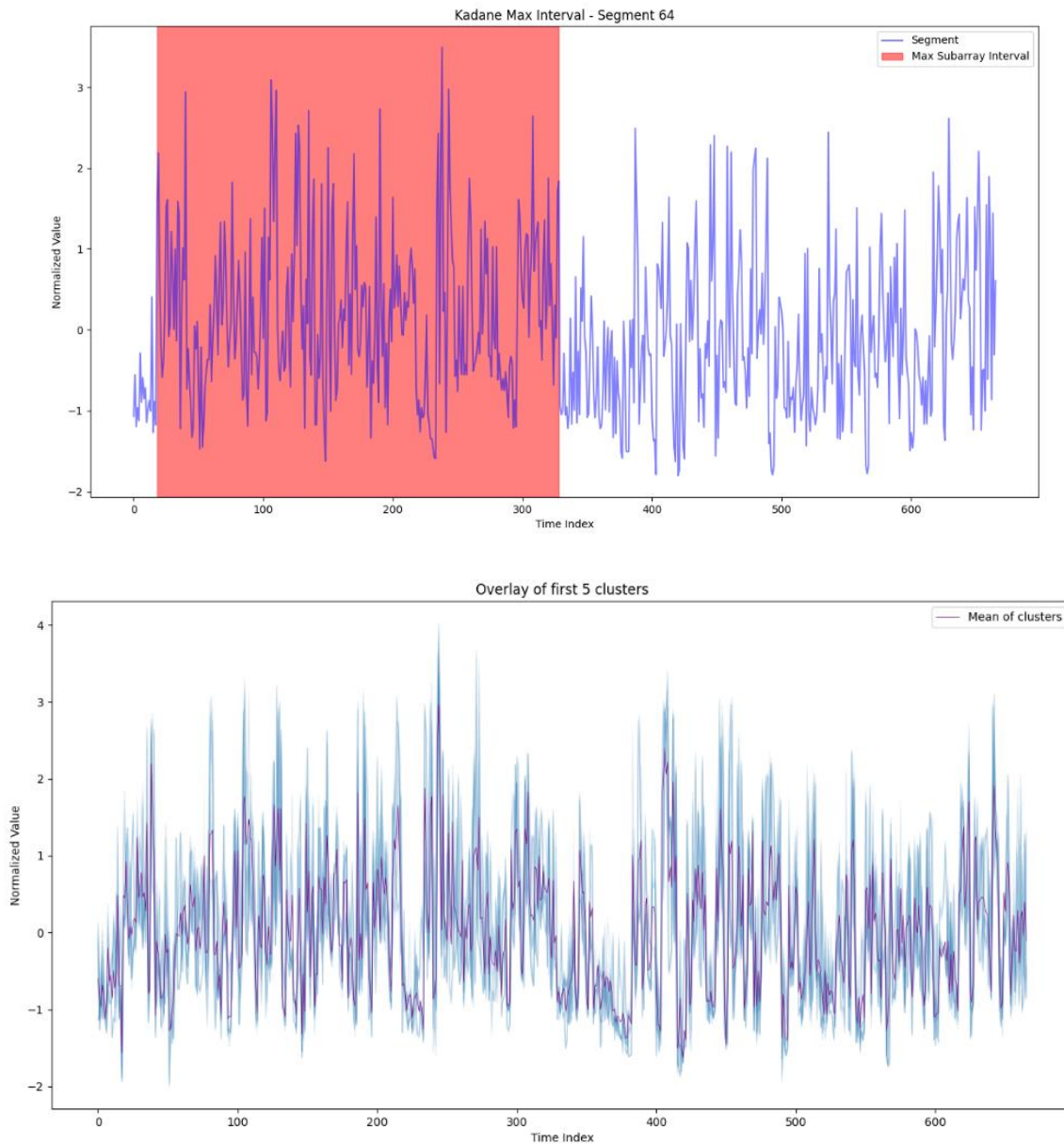
```
Visualizing closest pair of the first cluster...
```

```
Visualizing Kadane max interval for the first significant event segment...
```

```
@ts5100 -> /workspaces/GMP6462-DPOJECT1 - (main) 5
```







9. Execution Results

The execution of our pipeline demonstrates the effectiveness of divide-and-conquer clustering, dynamic time warping for closest pair detection, and Kadane's algorithm for identifying significant intervals in time-series segments.

Divide-and-Conquer Clustering: Using the ABP signals from the VitalDB subset, the algorithm produced 78 distinct clusters from 1,000 segments. The cluster sizes varied widely, from singleton clusters with only one segment to clusters containing up to 22

segments. For example, Cluster 1 contained 22 segments (indices 26–47), while Cluster 2 was a single segment (index 25). This variability highlights the algorithm’s ability to adaptively partition segments based on their correlation with the pivot centroid, effectively grouping highly similar segments while isolating outliers. The time complexity of this step is roughly $O(n^2 \cdot m)$, where n is the number of segments and m is the average segment length, due to pairwise correlation calculations during recursive splits.

Closest Pair Detection: Within each cluster, the DTW-based closest pair calculation identified the most similar pair of segments, capturing subtle shape similarities. For example, Cluster 1’s closest pair was segments 26 and 27 with a DTW distance of 0.161, while Cluster 30, containing a single segment, naturally had no closest pair. This step is computationally expensive with a worst-case complexity of $O(n^2 \cdot m_r)$, where m_r is the reduced segment length used for DTW distance computations.

Kadane’s Maximum Subarray Analysis: Kadane’s algorithm extracted the maximum subarray sum for each segment, providing insights into significant ABP fluctuations. Average maximum sums per cluster ranged widely, e.g., Cluster 1 had 57.953 and Cluster 35 reached 71.933. Start and end times were normalized for visualization in seconds, e.g., Cluster 1 avg_start = 1.08s, avg_end = 4.89s. The top 5% of significant events were highlighted, identifying 50 segments with extreme fluctuations, useful for clinical or anomaly detection analysis. Kadane’s algorithm operates in linear time $O(m)$ per segment, making it efficient even for large datasets.

Toy Example Verification: The toy example validated each algorithm in a controlled setting, producing two clusters, correctly identifying closest pairs within clusters, and computing Kadane intervals. Segment 0 yielded a max_sum of 1.903 at interval [4, 4], consistent with expected behavior.

10. Challenges

During the project, several challenges emerged:

1. **Cluster Imbalance:** The divide-and-conquer approach occasionally generated singleton clusters or very small clusters, especially when segments were outliers.

This required careful threshold tuning to balance cluster granularity with meaningful grouping.

2. **DTW Computation Cost:** Computing DTW distances for all segment pairs is computationally intensive, particularly for large clusters. Reducing segment size before DTW helped, but it introduces minor approximation errors.
3. **Segment Normalization:** Variability in baseline and amplitude across ABP signals necessitated normalization to prevent spurious clustering due to scale differences.
4. **Visualizing Overlapping Segments:** Overlaying multiple segments in plots required careful selection of line transparency and color palettes to maintain interpretability without clutter.
5. **Interpretation of Kadane Features:** While Kadane's algorithm efficiently identifies maximum subarrays, translating these into clinically meaningful events requires additional domain knowledge.

11. Conclusion

This project demonstrates a robust framework for time-series segment analysis using a combination of divide-and-conquer clustering, dynamic time warping, and Kadane's algorithm. The approach efficiently groups similar segments, identifies closest patterns within clusters, and highlights significant temporal intervals, providing actionable insights into the underlying signals.

The divide-and-conquer clustering proved flexible in handling heterogeneous data distributions, while DTW ensured precise similarity detection. Kadane's algorithm complemented these methods by pinpointing the most significant fluctuations. The toy example confirmed that each component functions correctly, and the ABP dataset analysis illustrated scalability and practical relevance.

Future improvements could include:

- Incorporating parallelization to accelerate DTW computations for large datasets.
- Adaptive thresholding for clustering to reduce singleton clusters and improve robustness.
- Integration with domain-specific event detection models to better interpret Kadane-identified intervals.

Overall, this project provides a clear, modular, and verifiable workflow for time-series analysis with strong potential for applications in physiological signal monitoring, anomaly detection, and trend analysis.