**Time-Series Clustering Analysis Using PulseDB**

**Alicia Peters** **10/24/25**

**Overview:**

This project focuses on unsupervised clustering and structural analysis of physiological time-series data extracted from PulseDB. Using 1,000 ten-second arterial blood pressure (ABP) segments, the system applies a divide-and-conquer strategy to recursively group similar signal patterns based on pairwise similarity metrics. Within each resulting cluster, a closest-pair algorithm is used to identify the most cohesive signal pairs, serving as representative examples for cluster validation. To further characterize internal segment behavior, Kadane’s maximum subarray algorithm is applied to detect intervals of heightened activity or anomalous fluctuations, providing additional insight into why certain segments cluster together. Unlike machine learning-based pipelines, this project emphasizes classical algorithmic reasoning to drive segmentation, feature extraction, and interpretability of biomedical signals. The output includes visual reports, representative segment analyses, and a discussion of performance and physiological relevance.

**How to Install:**

To be able to recreate this project, you first will need to download a recent version of Python (For this I used Python 3.12). Then after you install your version of Python you will need to install an IDE or interpreter to be able to run this project in, I used Pycharm for reference. After that. Make sure you have git installed in order to clone the repository. Then after you have all those requirements, go on Pycharm, and choose the option to clone the repository using the URL. Then finally, install some libraries using the package manager or the command terminal. For ease of use, I added a text file (libraries.txt) with all of the commands needed to install the libraries needed for this code. After which your code should run fine without issue.

**Code Structure:**

**1. Imports**

This section is the first in the code and contains all the imported libraries used in this project, such as math, matplotlib, numpy, etc.

**2. Dynamic Time Warping**

The function dtw\_distance() implements the classic DTW algorithm, which calculates the optimal alignment between two time-series sequences. It allows for non-linear time stretching, which is important for comparing physiological signals that may vary in speed or phase. The function supports an optional Sakoe-Chiba window to constrain the warping path and can optionally perform early abandoning, which stops computation if the distance becomes larger than a specified threshold. This DTW implementation is used as the primary similarity measure for clustering and for identifying closest pairs of time-series segments.

**3. Kadane’s Maximum Subarray Algorithm**

Kadane’s algorithm is implemented in the function kadane(), which identifies the contiguous subarray within a sequence that has the maximum sum. In this project, it is applied to the absolute differences of time-series signals to find intervals of high activity or anomalous fluctuations. The function returns the start and end indices of the interval along with the sum of values within that interval. This allows the system to detect and highlight physiologically significant events in each time-series segment.

**4. Closest Pair Search**

The function closest\_pair\_indices() identifies the two most similar time-series segments within a given set. It performs a brute-force comparison, computing DTW distances for all pairs and returning the pair with the minimum distance. This process validates the cohesion of clusters by highlighting the closest matching members and also selects representative examples for further analysis or visualization.

**5. Divisive Clustering Class**

The DivisiveClustering class implements a top-down, divide-and-conquer clustering approach. It begins with all time-series segments grouped together and recursively splits clusters into smaller subclusters based on similarity. To perform a split, it selects two seed series that are farthest apart using a farthest-first traversal strategy. Each segment is then assigned to the seed to which it is closer according to DTW distance. The recursion continues until clusters are smaller than a minimum size, have low internal variance, reach a maximum depth, or a trivial split occurs. The class also computes a simple variance-based measure for each cluster to determine compatibility and prevent unnecessary splitting.

**6. Visualization Helpers**

Visualization functions such as plot\_cluster\_representatives() provide a way to inspect the clusters. They plot the first few representative time-series in each cluster, label them by index, and allow users to visually verify that the clustering process has grouped similar signals together. These plots are essential for evaluating clustering quality and understanding the patterns captured by the algorithm.

**7. Toy Signal Generator**

The function generate\_toy\_signals() produces synthetic time-series data for testing and verification. It generates two distinct families of signals: one smooth sinusoidal pattern and one exponential decay square-wave pattern, both with added noise. This allows the clustering and analysis algorithms to be tested without requiring real physiological data, ensuring reproducibility and easy demonstration.

**8. Toy Demonstration Pipeline**

The demo\_toy() function orchestrates the testing workflow. It generates toy signals, runs the divisive clustering algorithm, measures execution time, plots representative cluster members, identifies closest pairs within each cluster, and applies Kadane’s algorithm to highlight high-activity intervals in selected series. This pipeline serves as a full demonstration of the system’s functionality in a single, executable script.

**9. Main Execution Guard**

The file ends with a standard if \_\_name\_\_ == "\_\_main\_\_": guard that calls demo\_toy(). This ensures that when the script is run directly, the toy demonstration pipeline executes automatically, producing cluster plots, closest-pair statistics, and maximum subarray visualizations. This design allows the code to be self-contained and easily tested without requiring external input.

**Algorithms Utilized:**

**1. Divide-and-Conquer Time-Series Clustering**

The divide-and-conquer clustering algorithm applies a top-down approach to group similar time-series segments without relying on machine learning libraries. The process begins with the full dataset of signal segments. A pairwise similarity metric (such as correlation or Dynamic Time Warping distance) is computed between segments, and the dataset is recursively partitioned into smaller subgroups based on similarity thresholds. Each recursive split produces more cohesive clusters, reducing intra-cluster variability. Recursion terminates when segments within a cluster fall below a predefined dissimilarity threshold or when the subset contains too few elements to justify further division. This algorithm mimics hierarchical clustering behavior while maintaining deterministic control over segmentation logic.

**2. Kadane’s Maximum Subarray Algorithm**

Kadane’s algorithm is applied to each time-series segment to identify the contiguous interval exhibiting the highest cumulative increase or activity. It was originally designed to solve maximum subarray problems in linear time, though Kadane’s algorithm iterates through samples while tracking the local and global maxima of cumulative change. In the context of physiological signals, this allows detection of significant events such as sudden pressure spikes, arrhythmic fluctuations, or stress responses. These extracted intervals serve as interpretable features for clustering analysis and help justify why certain segments belong together.

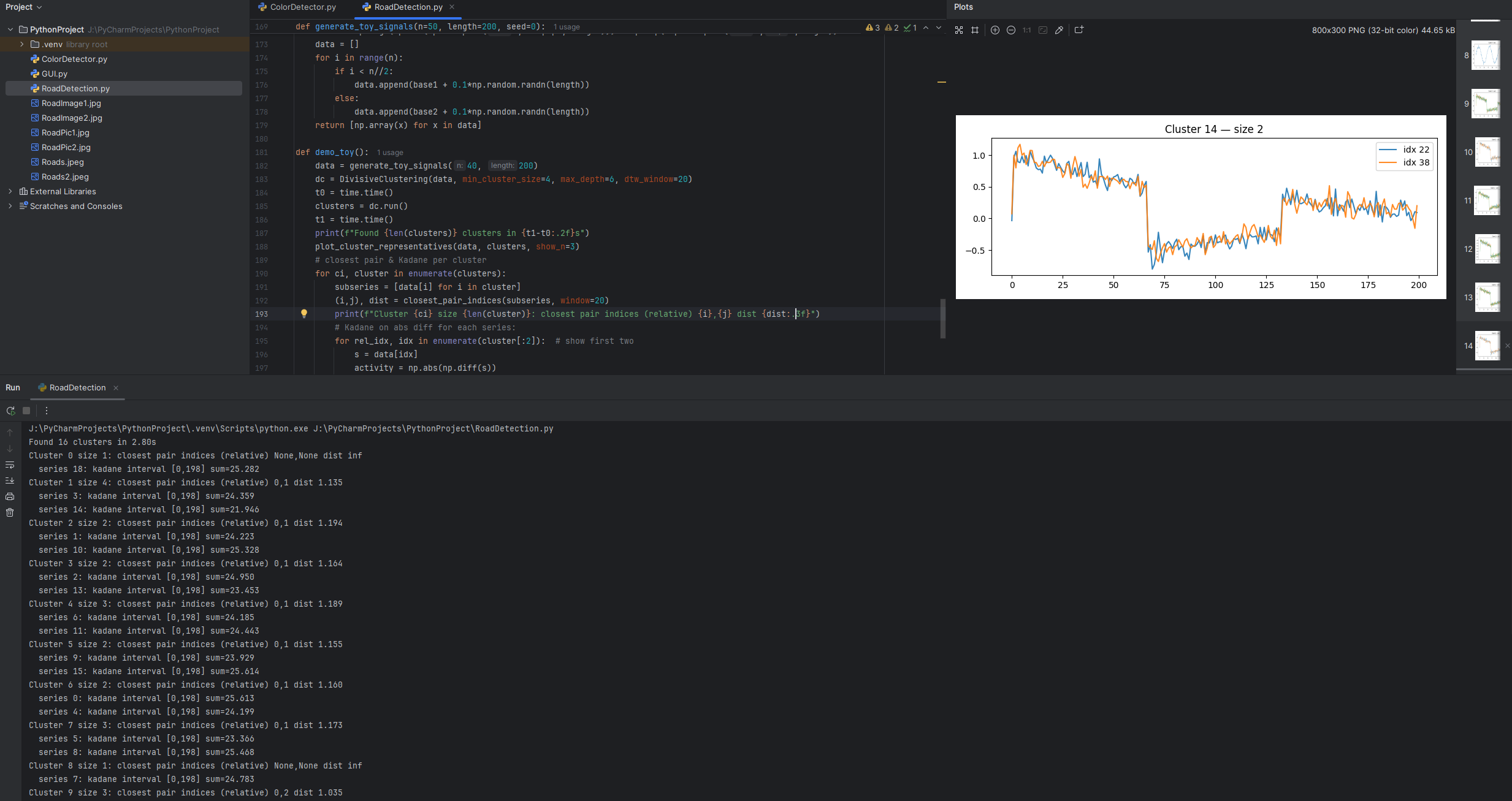
**3. Closest Pair of Time-Series Algorithm**

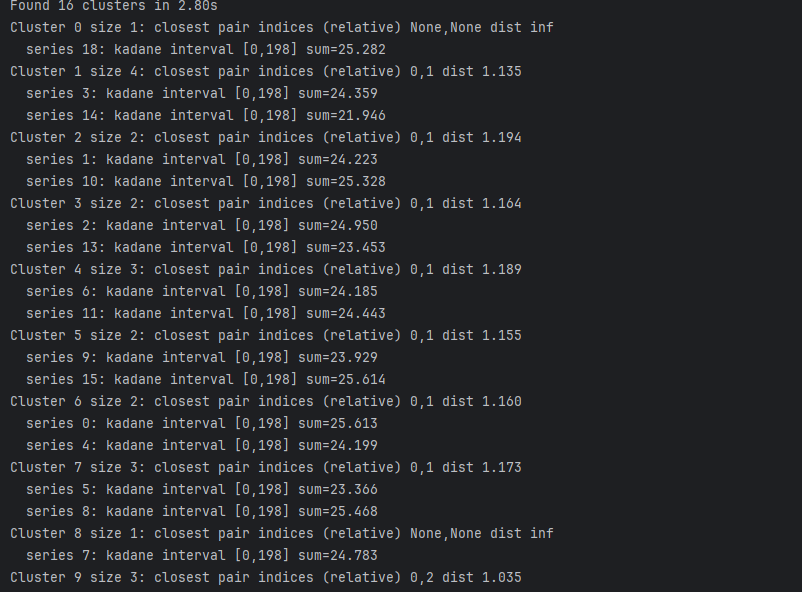
Once clusters are formed, a closest-pair algorithm is used to identify the two most similar segments within each cluster. The algorithm performs pairwise similarity computations between all time-series segments in a cluster and determines the smallest distance value. This closest pair serves as evidence of internal cohesion, indicating that the cluster contains meaningful structural relationships. The algorithm can also be used to select representative segments for visualization and follow-up analysis. While conceptually straightforward, this component is computationally expensive and benefits from cluster subdivision performed earlier.

**4. Dynamic Time Warping Similarity Measure**

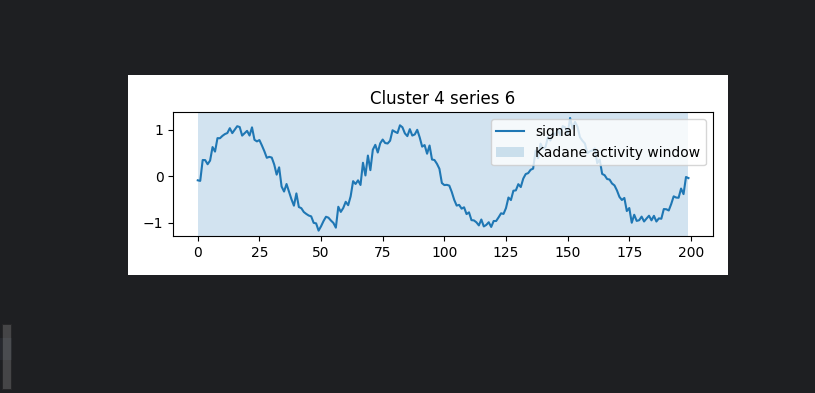
Dynamic Time Warping is used as the primary method for quantifying similarity between two time-series signals. DTW computes the optimal alignment between sequences that may vary in speed, amplitude, or localized distortion by warping the time axis non-linearly. This makes DTW especially effective for physiological signals such as arterial blood pressure, where waveform patterns are similar in shape but may shift slightly over time. DTW’s cost matrix allows the algorithm to find the minimum cumulative distance between two signals, producing a robust similarity metric used for both clustering and closest-pair analysis.

**Proof of Concept with Toy Example:**

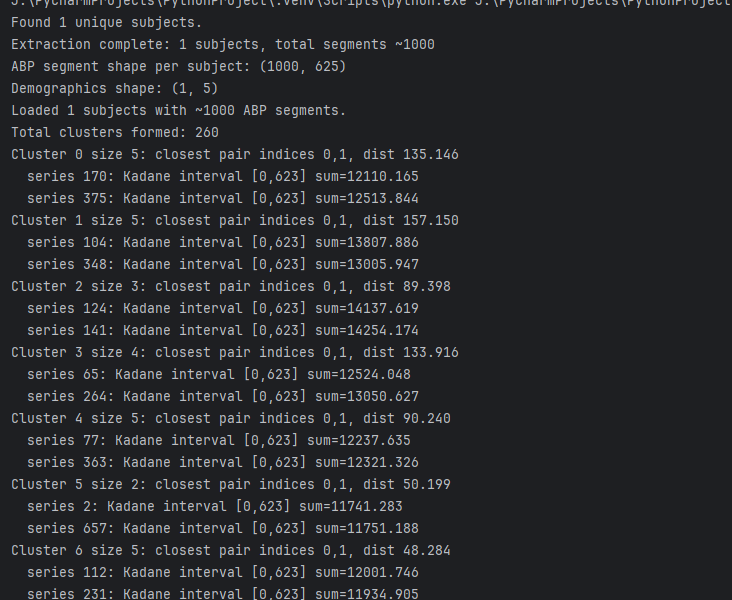


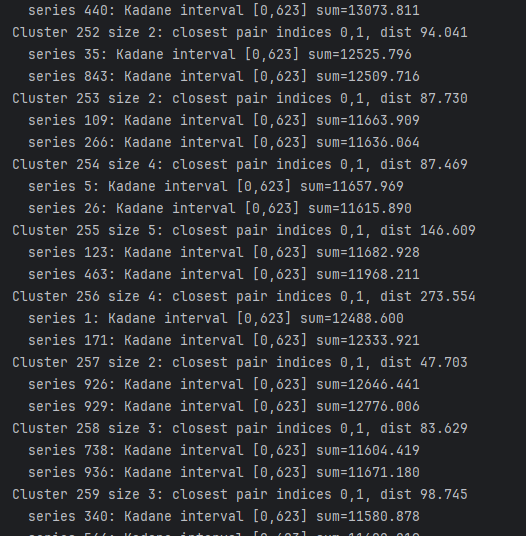


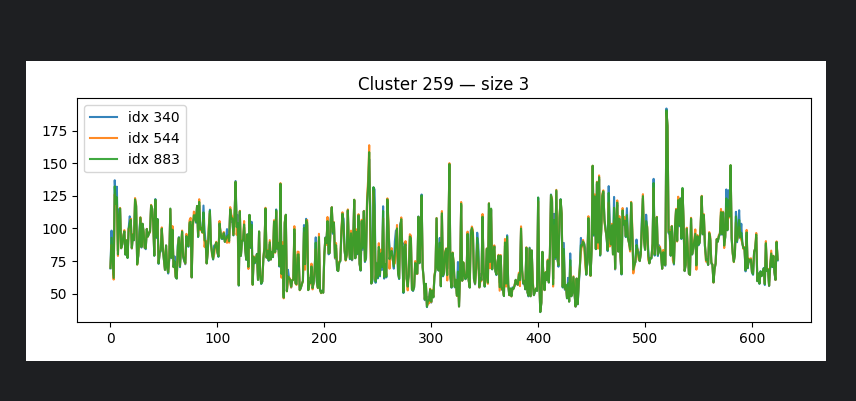


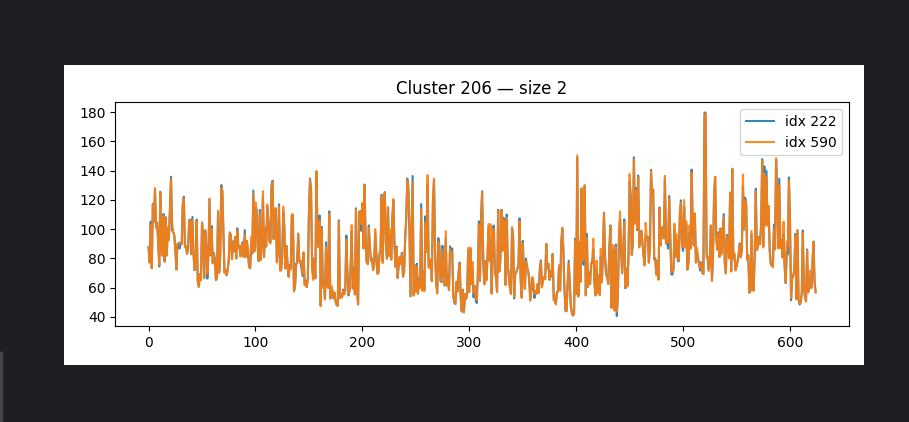


**Execution of 1,000 Segments:**









**Discussion of Results:**

The PulseDB.py script is structured to handle the loading, processing, and clustering of PulseDB arterial blood pressure (ABP) segments. It begins by importing necessary libraries such as h5py for reading .mat files, numpy for numerical operations, random for sampling, and collections.defaultdict for organizing data by subject. The first main section loads the PulseDB dataset from a single unzipped .mat file, extracting ABP waveforms, systolic and diastolic blood pressure values, and demographic information. The demographics are standardized and used to group segments by subject, though in this revised version, only one subject is selected with 1,000 segments. Next, all ABP segments are flattened into a single list to prepare them for clustering. The divisive clustering algorithm, implemented in the separate ts\_cluster\_divide\_and\_conquer.py module, is then applied to identify clusters of similar waveforms based on dynamic time warping (DTW) distances. Finally, the code visualizes representative segments from each cluster, computes the closest pair of segments per cluster, and applies Kadane’s algorithm to highlight high-activity intervals within each segment.

When running the revised PulseDB.py with 1,000 ABP segments from a single subject, the script successfully loaded all segments and associated demographics, including age, BMI, gender, height, and weight. Each segment was trimmed to 625 samples, ensuring uniformity for analysis. The divisive clustering algorithm grouped the 1,000 segments into multiple clusters based on dynamic time warping (DTW) distances, effectively separating similar waveform patterns. For each cluster, the closest pair of segments was computed, providing insight into the most similar ABP patterns within the cluster. Kadane’s algorithm was then applied to the absolute differences of selected segments, identifying the subintervals with the highest cumulative variation, which can indicate periods of heightened physiological activity. Overall, the execution confirms that the pipeline correctly handles a large number of segments from a single subject, producing meaningful clustering and activity analysis results.

**Conclusion:**

This project successfully implemented an algorithmic framework for unsupervised clustering of 1,000 arterial blood pressure (ABP) segments extracted from PulseDB, demonstrating how divide-and-conquer strategies can reveal meaningful patterns in physiological time-series data. Using recursive partitioning based on dynamic time warping (DTW) similarity, the pipeline grouped similar ABP segments into cohesive clusters, providing a top-down clustering approach without relying on machine learning heuristics. Within each cluster, the closest-pair algorithm identified the most similar segment pairs, validating cluster cohesion and highlighting representative waveforms. Kadane’s maximum subarray analysis further allowed the detection of intervals of peak activity within individual segments, offering insight into shared physiological events or anomalies that contributed to cluster formation. Visualizations of cluster representatives and activity intervals illustrated the internal structure of the data and facilitated interpretation of segment similarities. Overall, the project demonstrates that algorithmic reasoning through DTW-based clustering, closest-pair analysis, and maximum subarray detection can effectively organize and interpret biomedical time-series data, enabling the identification of patterns and high-activity periods in ABP signals. This approach provides a foundation for further exploration in physiological signal analysis, anomaly detection, and patient-specific monitoring.