Time-Series Clustering and Segment Analysis on PulseDB Using Divide-and-Conquer Algorithms

Daisy Aptovska

Computer Science Program, The Pennsylvania State University

CMPSC 463: Design and Analysis of Algorithms

Dr. Janghoon Yang

October 25, 2025

Table of Contents

Project Description

In this project, three algorithms are executed on the PulseDB dataset. The dataset is analyzed for different types of pulse rate measures such as ABP. Through an algorithmic approach, this project clusters the data through leveraging Euclidean distance and K-Means, displaying how many clusters there are and how many time segments are in each cluster. Once the data is clustered, the project finds the closet pair of data points

Installation & Usage

To conduct this project, multiple installations are needed. Firstly, a version of Python 3.x should be installed on the device (unless a cloud-based notebook is being used). This implementation used Python 3.13.2. Then, within the Python installation, an installation of multiple packages is needed. Firstly, NumPy is needed to efficiently process the dataset as NumPy arrays. This implementation used NumPy 2.2.4. Secondly, the h5py package is used to process the data file (the PulseDB dataset .mat file) in a HDF5 binary data format. This provides efficiency for processing large amounts of data, and it works seamlessly with the NumPy library for data manipulation. This implementation used h5py 3.15.1.

Once Python is properly installed with the previously mentioned packages, the dataset needs to be downloaded to run the program properly. The PulseDB dataset can be downloaded from the Kaggle website or from the GitHub repository. Due to the large size of this dataset, this implementation uses 1000 time-series segments from 1 test subject. However, this dataset can be scaled for a much larger implementation, testing, comparing, and analyzing hundreds of subjects' time-series segments data. This implementation used a downloaded file, the "VitalDB_CalBased_Test_Subset.mat" file from the Kaggle website within the PulseDB dataset, to read and process the data. After the Python, Python packages, and dataset installation, the three project files can be downloaded, and the main file (project1_1.py) can be ran and executed.

The usage of this project is to see an analysis for a PulseDB dataset subject and 1000 of their time-series segments for their ABP. This analysis provides a clustering of the ABPs recorded for the subject, identifies the closest pairs of time-series segments within each cluster, and applies Kadane's algorithm to each time-series.

Code Structure

The code structure of this project follows a logical structure. There are three Python files within this project along with one dataset (.mat) file. In project1_1.py, the main function is present to run the project. Along with the main function, this file reads, parses, and provides summary statistics of the data in the file. After parsing the data and getting all the ABP, SBP, and DBP segments, the program goes into the cluster data function. Here, the program finds the Euclidean distances between each time-series segment in the ABP segments array. Once all the Euclidean distances between all time-series are found, they are put into a Euclidean distance matrix of 1000 by 1000. With the distance matrix, the data can be clustered and plotted through a manual K-Means clustering method (LeCheminant, 2020).

Based on this visual output, there are 100 clusters in the 1000 segments for this subject. Additionally, the code was tested with 50 clusters, but the yielded results were not as consistent. When the k value was initialized as a smaller value, they segments tended to cluster into 1-3 clusters, resulting in inconsistencies since all the data was focused into few clusters, leaving more clusters empty than with a larger k value.

Lastly, in the third and last portion of this project, Kadane's algorithm is applied to each time segment. Here, the maximum subarray sum is calculated within each time segment. By iterating over every element and adding each element after it to itself, the sum is found, and it is compared with the previous elements result. Once the maximum between the two maximums is found, it is returned as the maximum sum within the array from a subarray.

Algorithm Description

In this project, there were three algorithms applied to yield these results. The first algorithm applied is to cluster the data. Clustering the data is applied in project1_1.py, using the "clusterData" function. Here, the Euclidean distance of each time series segment is found between itself and every other time series segment. From this result, the program returns a Euclidean distance matrix, comparing all the time series segments to one another and their distances. From this matrix, the clusters are calculated by comparing these distances. Then, the clusters are recalculated after initializing the clusters as individual data points (LeCheminant, 2020). This follows a manual K-Means approach by using a k value of 100 (and 50 as a second text case). Then, the final clusters are found.

In the second algorithm, the closest pairs within each non-empty cluster are found. The closest pairs represent the minimum distance between 2 time-series segments in each cluster. In this algorithm, each cluster is iterated through, comparing each time-series segment with every other time-series segment in the cluster. They are compared by calculating the Euclidean distance between each time-series segment. From here, the distance is added to a list of distances. From this list of distances, the minimum is found. Then, the minimum of each cluster can be found. Lastly, the function returns the cluster, the distance between the closest pair, and the first and second segments in the pair.

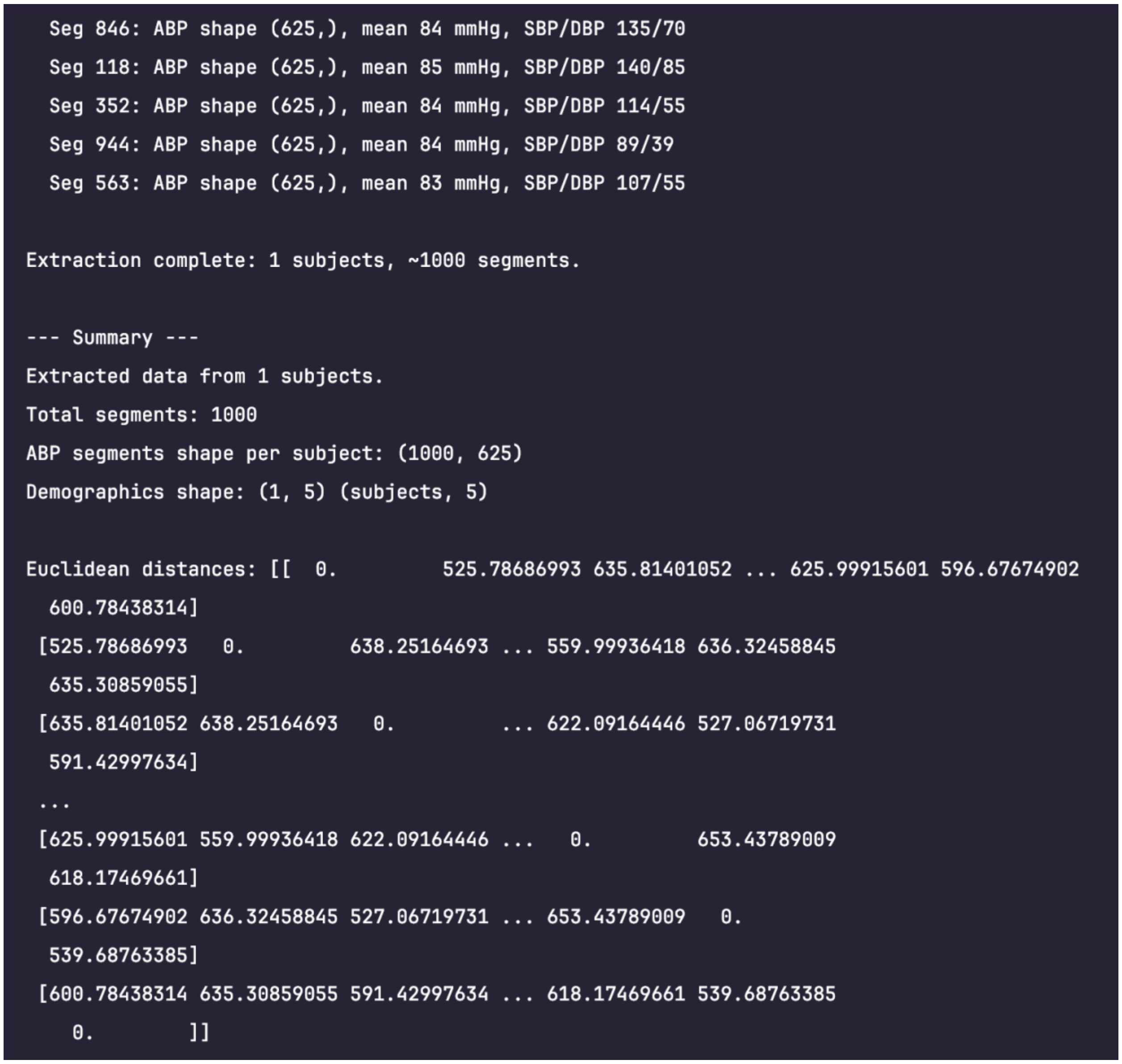
In the third algorithm, Kadane's algorithm is implemented. Kadane's algorithm is an algorithm that finds the maximum sum within a subarray of an array (Kartik, 2025a). In this function, the ABP time-series segments pulse data is passed. For each of the 1000 time-series segments, the maximum is found by iterating through the entire segment. A running sum count is tracked and compared to the beginning of the array. Then, the maximum of the 2 values is returned, ultimately returning the maximum subarray sum within the time-series segment.

Functionality Verification

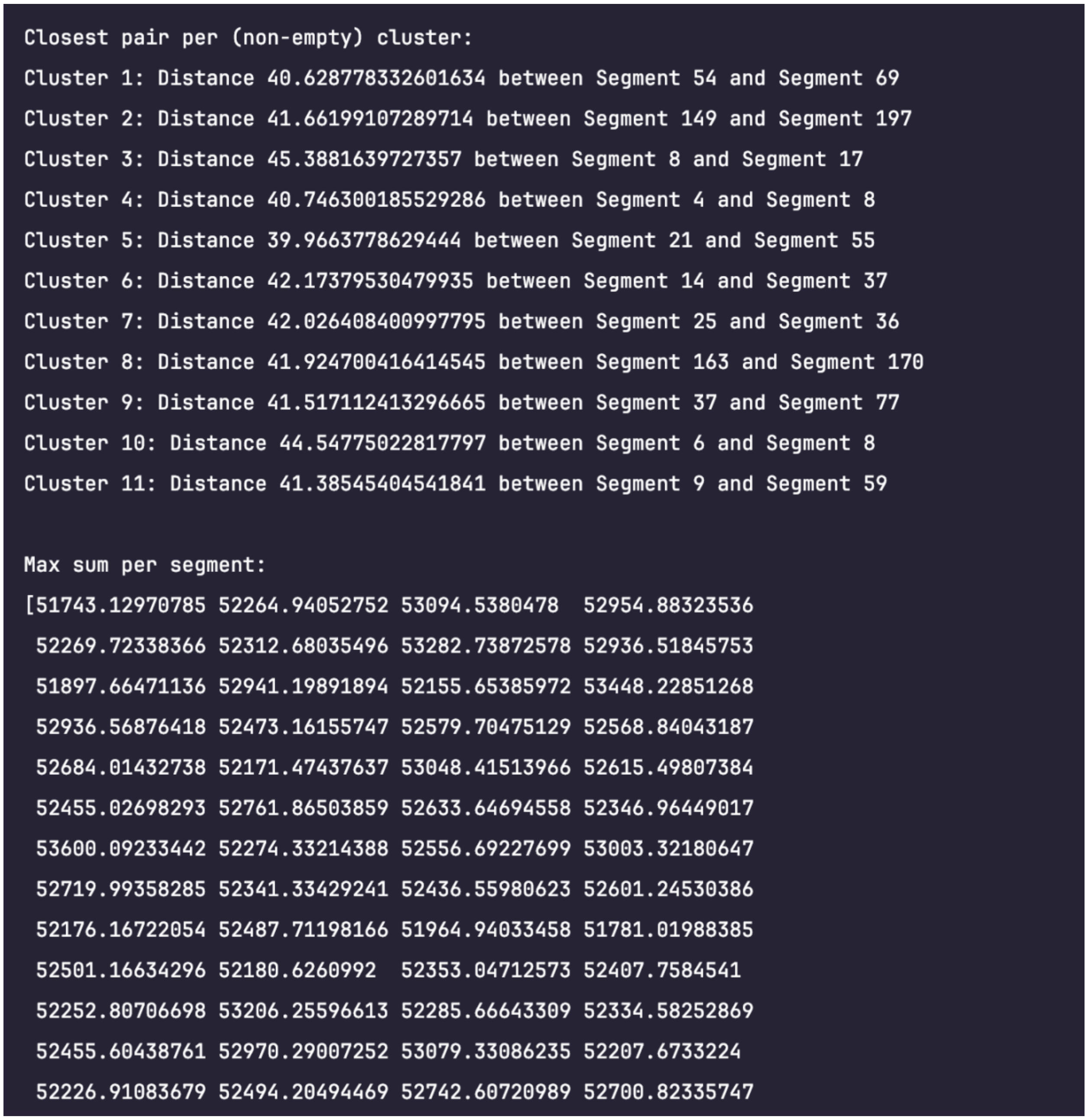
Based on the development of this project, this analysis can be applied to a variety of datasets and analyses. Other medical analyses could be compared through this project. Patients could be compared to the clustered data to see how different or similar their pulse data is. Additionally, different pulse measurements could be compared with this project. This project is useful in the medical field to check for outliers against a large dataset of other patient medical data. It can be used to determine if patients need care based on their pulse and if they are susceptible to any illnesses.

Execution Results (1000 segments, 1 subject)

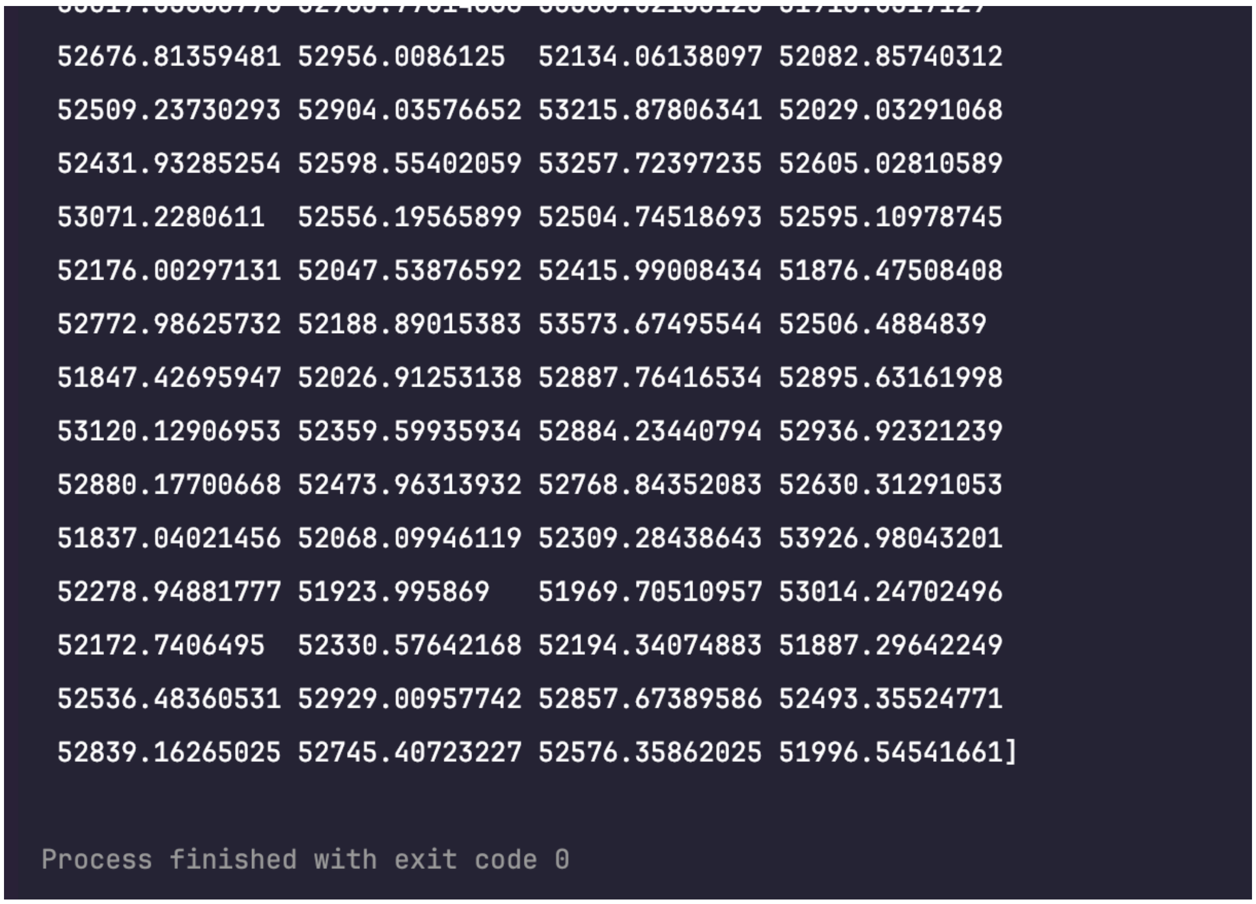
The following snapshots show the output of the project with using a k value for K-means of 100 (100 clusters).



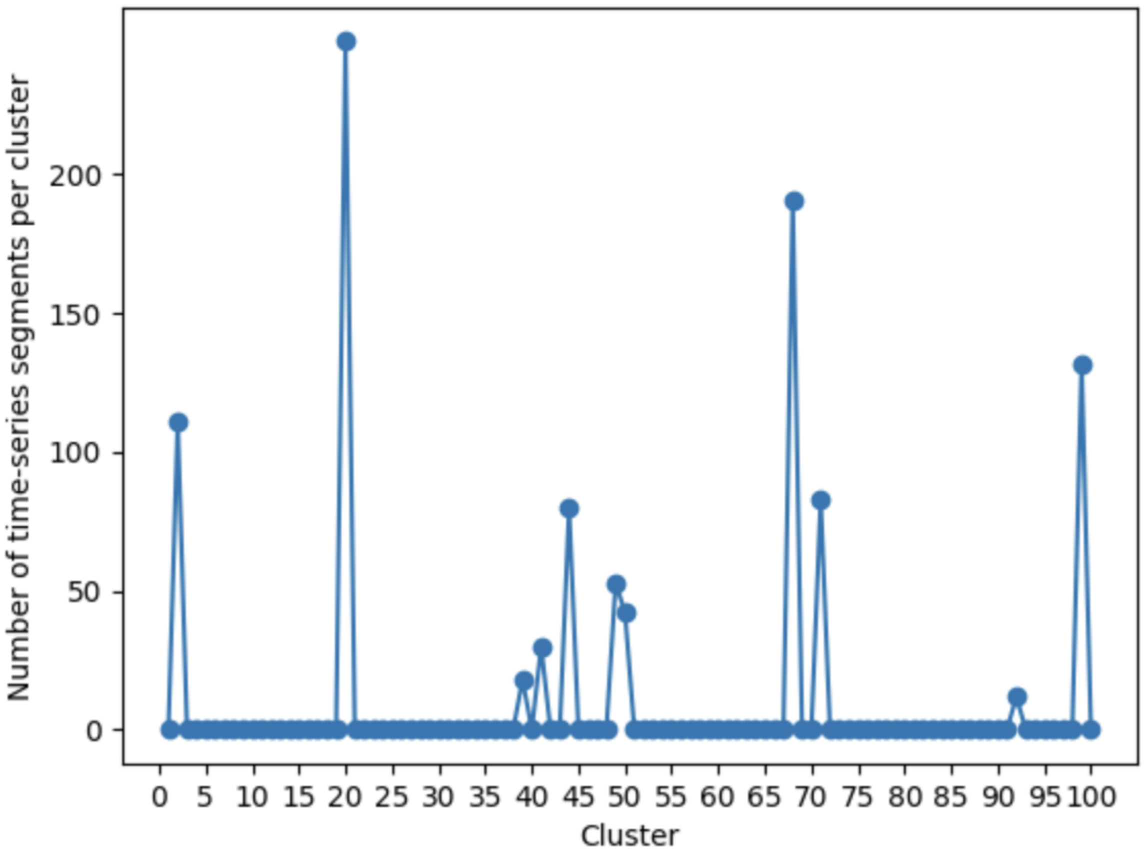
```
  Seg 846: ABP shape (625,), mean 84 mmHg, SBP/DBP 135/70
  Seg 118: ABP shape (625,), mean 85 mmHg, SBP/DBP 140/85
  Seg 352: ABP shape (625,), mean 84 mmHg, SBP/DBP 114/55
  Seg 944: ABP shape (625,), mean 84 mmHg, SBP/DBP 89/39
  Seg 563: ABP shape (625,), mean 83 mmHg, SBP/DBP 107/55

Extraction complete: 1 subjects, ~1000 segments.

--- Summary ---
Extracted data from 1 subjects.
Total segments: 1000
ABP segments shape per subject: (1000, 625)
Demographics shape: (1, 5) (subjects, 5)

Euclidean distances: [[  0.         525.78686993 635.81401052 ... 625.99915601 596.67674902
  600.78438314]
 [525.78686993   0.         638.25164693 ... 559.99936418 636.32458845
  635.30859055]
 [635.81401052 638.25164693   0.         ... 622.09164446 527.06719731
  591.42997634]
 ...
 [625.99915601 559.99936418 622.09164446 ...   0.         653.43789009
  618.17469661]
 [596.67674902 636.32458845 527.06719731 ... 653.43789009   0.
  539.68763385]
 [600.78438314 635.30859055 591.42997634 ... 618.17469661 539.68763385
    0.        ]]
```

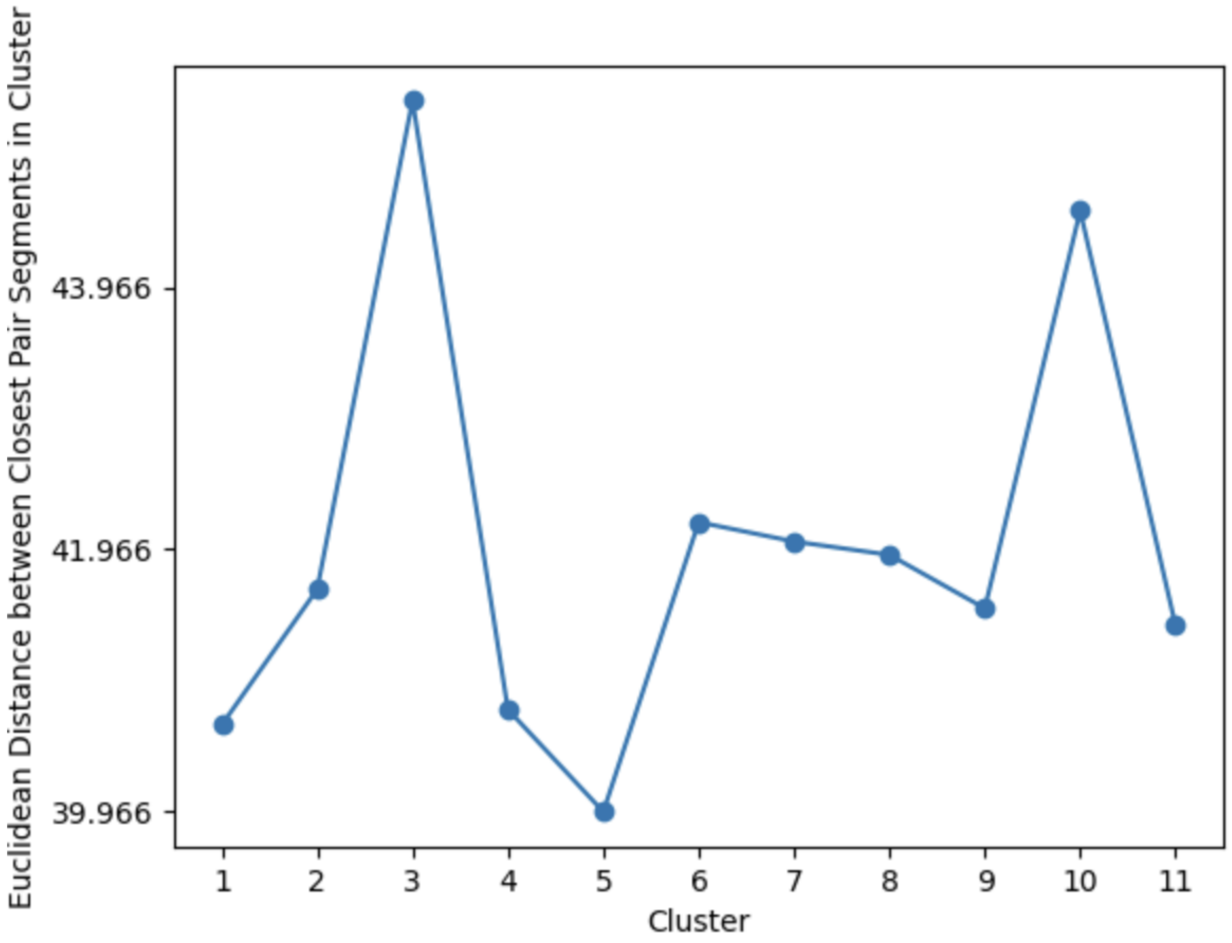


```
Closest pair per (non-empty) cluster:
Cluster 1: Distance 40.628778332601634 between Segment 54 and Segment 69
Cluster 2: Distance 41.66199107289714 between Segment 149 and Segment 197
Cluster 3: Distance 45.3881639727357 between Segment 8 and Segment 17
Cluster 4: Distance 40.746300185529286 between Segment 4 and Segment 8
Cluster 5: Distance 39.9663778629444 between Segment 21 and Segment 55
Cluster 6: Distance 42.17379530479935 between Segment 14 and Segment 37
Cluster 7: Distance 42.026408400997795 between Segment 25 and Segment 36
Cluster 8: Distance 41.924700416414545 between Segment 163 and Segment 170
Cluster 9: Distance 41.517112413296665 between Segment 37 and Segment 77
Cluster 10: Distance 44.54775022817797 between Segment 6 and Segment 8
Cluster 11: Distance 41.38545404541841 between Segment 9 and Segment 59

Max sum per segment:
[51743.12970785 52264.94052752 53094.5380478  52954.88323536
 52269.72338366 52312.68035496 53282.73872578 52936.51845753
 51897.66471136 52941.19891894 52155.65385972 53448.22851268
 52936.56876418 52473.16155747 52579.70475129 52568.84043187
 52684.01432738 52171.47437637 53048.41513966 52615.49807384
 52455.02698293 52761.86503859 52633.64694558 52346.96449017
 53600.09233442 52274.33214388 52556.69227699 53003.32180647
 52719.99358285 52341.33429241 52436.55980623 52601.24530386
 52176.16722054 52487.71198166 51964.94033458 51781.01988385
 52501.16634296 52180.6260992  52353.04712573 52407.7584541
 52252.80706698 53206.25596613 52285.66643309 52334.58252869
 52455.60438761 52970.29007252 53079.33086235 52207.6733224
 52226.91083679 52494.20494469 52742.60720989 52700.82335747
```

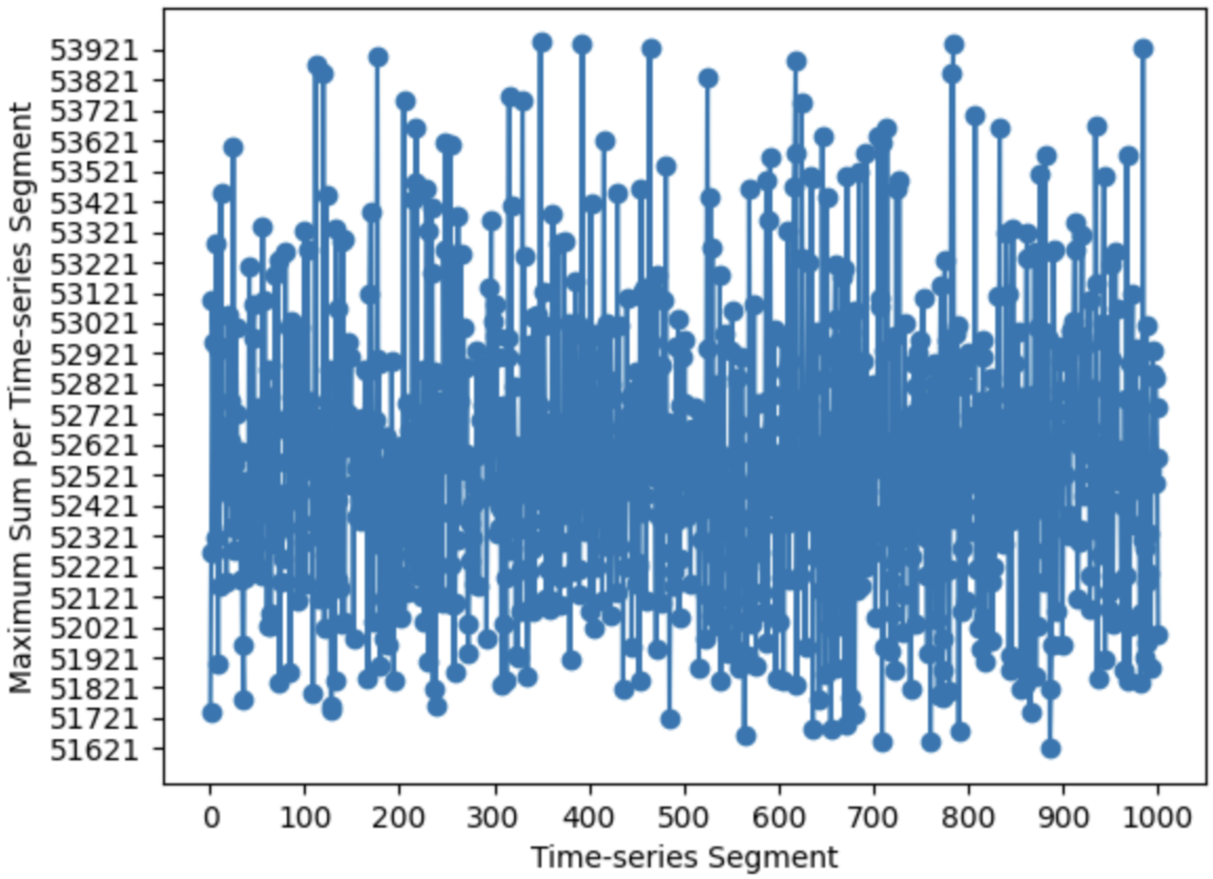


```
 [illegible]
 52676.81359481 52956.0086125  52134.06138097 52082.85740312
 52509.23730293 52904.03576652 53215.87806341 52029.03291068
 52431.93285254 52598.55402059 53257.72397235 52605.02810589
 53071.2280611  52556.19565899 52504.74518693 52595.10978745
 52176.00297131 52047.53876592 52415.99008434 51876.47508408
 52772.98625732 52188.89015383 53573.67495544 52506.4884839
 51847.42695947 52026.91253138 52887.76416534 52895.63161998
 53120.12906953 52359.59935934 52884.23440794 52936.92321239
 52880.17700668 52473.96313932 52768.84352083 52630.31291053
 51837.04021456 52068.09946119 52309.28438643 53926.98043201
 52278.94881777 51923.995869   51969.70510957 53014.24702496
 52172.7406495  52330.57642168 52194.34074883 51887.29642249
 52536.48360531 52929.00957742 52857.67389586 52493.35524771
 52839.16265025 52745.40723227 52576.35862025 51996.54541661]

Process finished with exit code 0
```

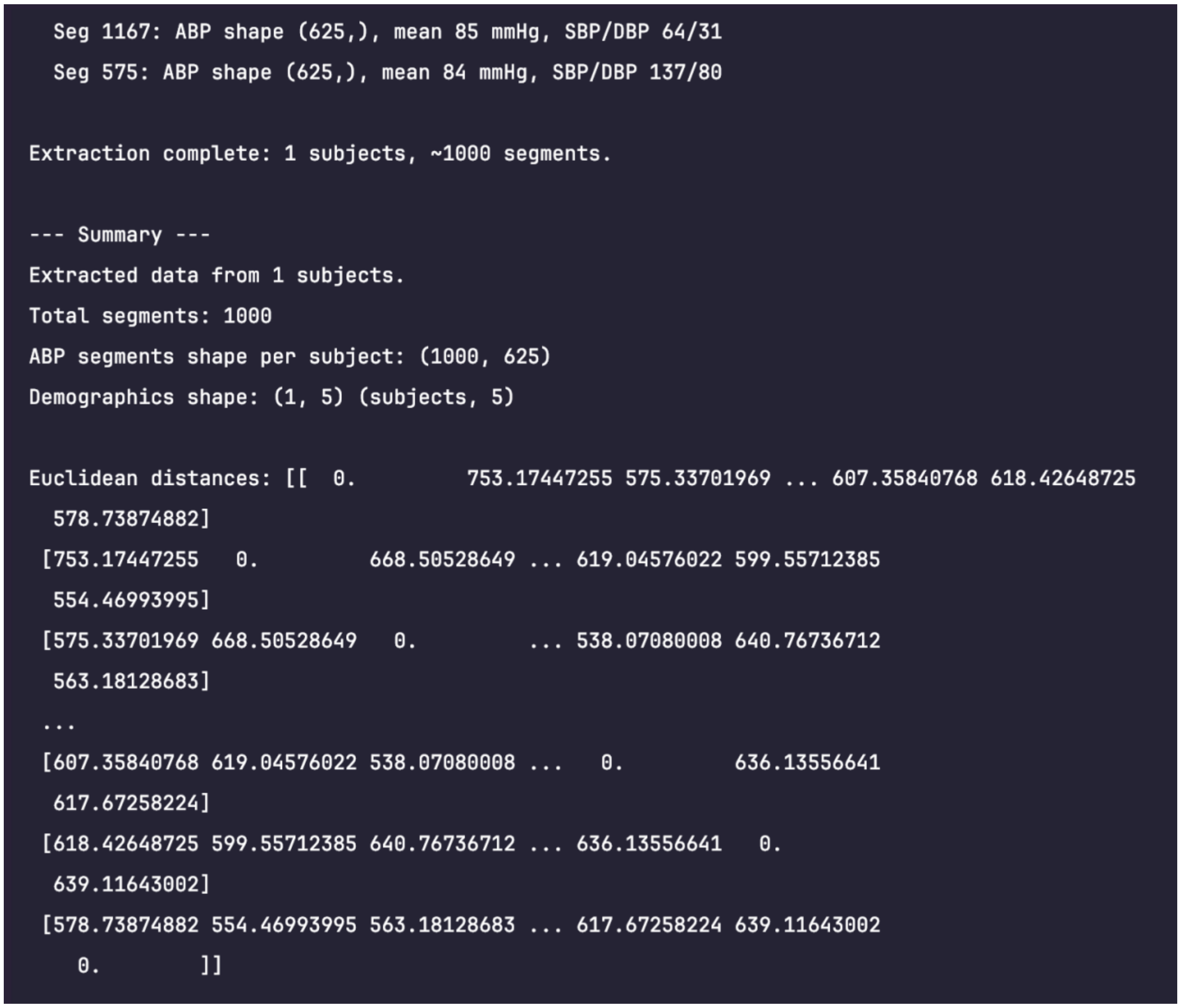

Number of time-series segments per cluster
0
50
100
150
200
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100
Cluster



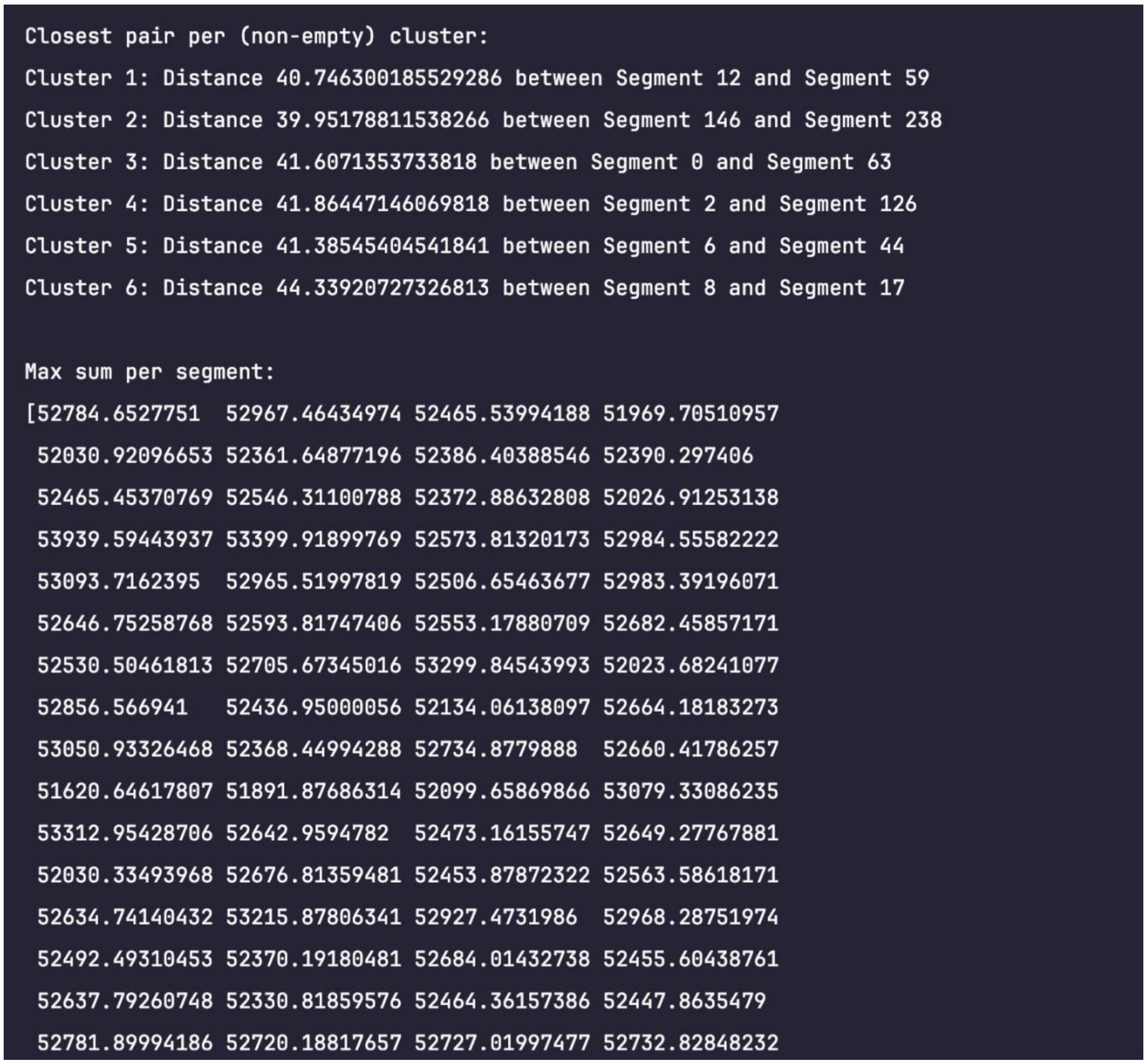
Euclidean Distance between Closest Pair Segments in Cluster
43.966
41.966
39.966
1
2
3
4
5
6
7
8
9
10
11
Cluster



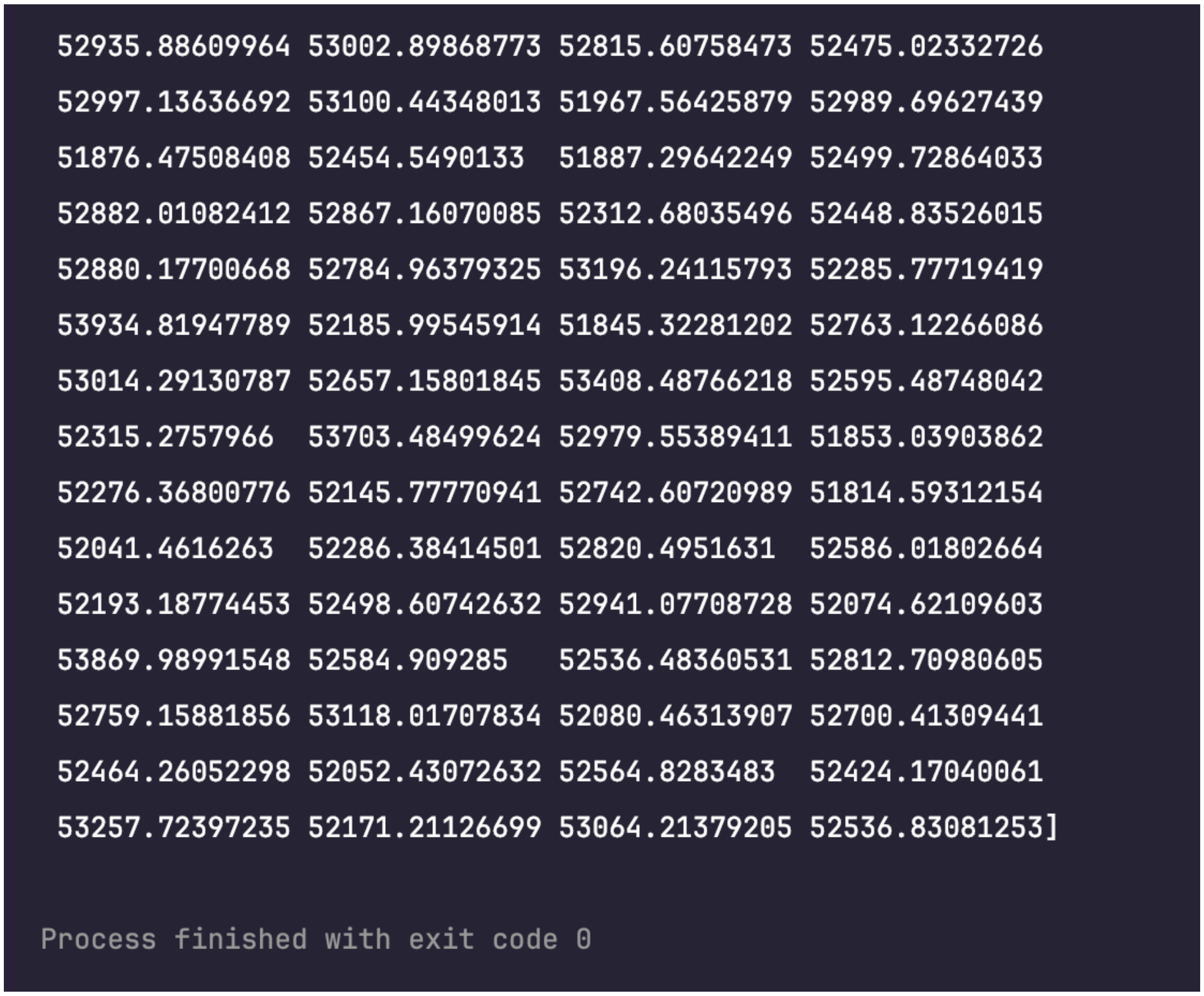
The following snapshots show the output of the project with using a k value for K-means of 50 (50 clusters).



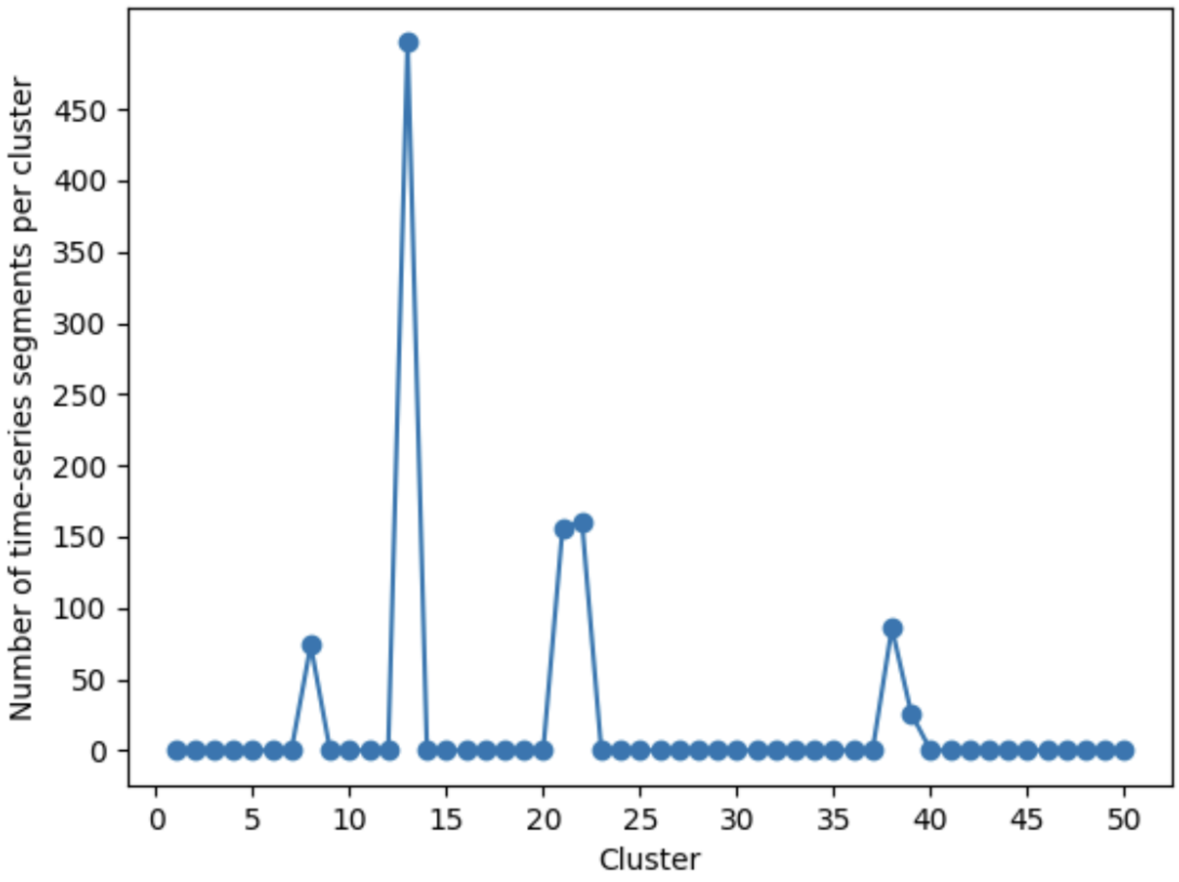
```
  Seg 1167: ABP shape (625,), mean 85 mmHg, SBP/DBP 64/31
  Seg 575: ABP shape (625,), mean 84 mmHg, SBP/DBP 137/80

Extraction complete: 1 subjects, ~1000 segments.

--- Summary ---
Extracted data from 1 subjects.
Total segments: 1000
ABP segments shape per subject: (1000, 625)
Demographics shape: (1, 5) (subjects, 5)

Euclidean distances: [[  0.         753.17447255 575.33701969 ... 607.35840768 618.42648725
  578.73874882]
 [753.17447255   0.         668.50528649 ... 619.04576022 599.55712385
  554.46993995]
 [575.33701969 668.50528649   0.         ... 538.07080008 640.76736712
  563.18128683]
 ...
 [607.35840768 619.04576022 538.07080008 ...   0.         636.13556641
  617.67258224]
 [618.42648725 599.55712385 640.76736712 ... 636.13556641   0.
  639.11643002]
 [578.73874882 554.46993995 563.18128683 ... 617.67258224 639.11643002
    0.        ]]
```

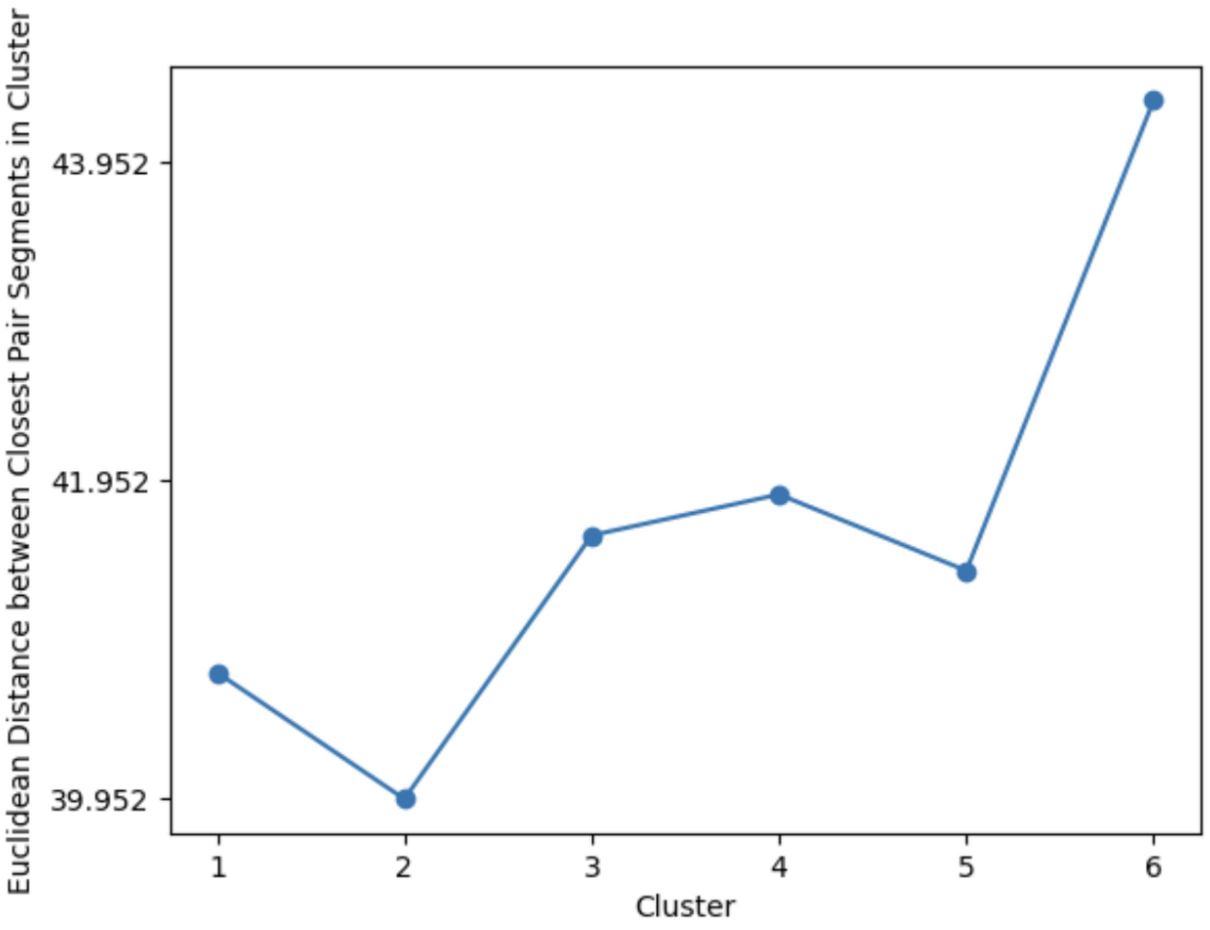


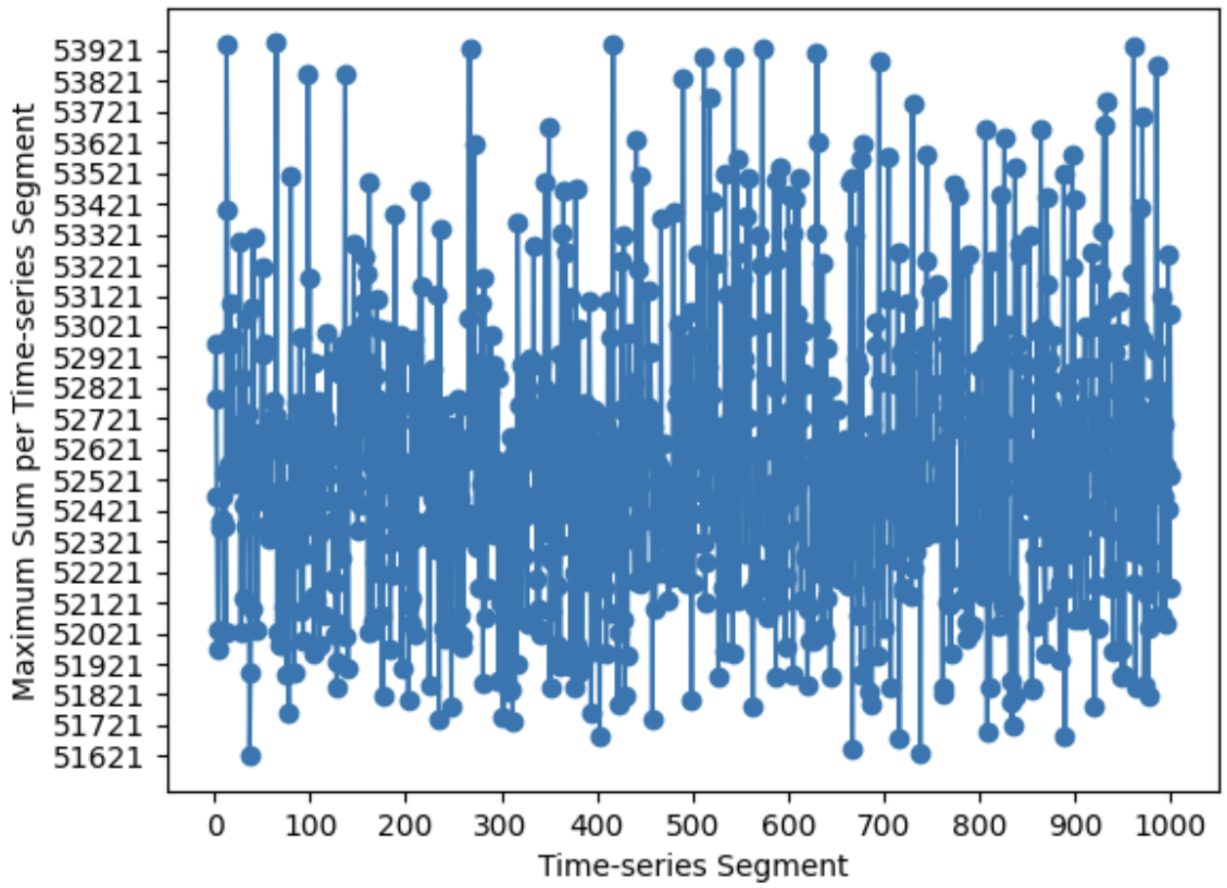
```
Closest pair per (non-empty) cluster:
Cluster 1: Distance 40.746300185529286 between Segment 12 and Segment 59
Cluster 2: Distance 39.95178811538266 between Segment 146 and Segment 238
Cluster 3: Distance 41.6071353733818 between Segment 0 and Segment 63
Cluster 4: Distance 41.86447146069818 between Segment 2 and Segment 126
Cluster 5: Distance 41.38545404541841 between Segment 6 and Segment 44
Cluster 6: Distance 44.33920727326813 between Segment 8 and Segment 17

Max sum per segment:
[52784.6527751  52967.46434974 52465.53994188 51969.70510957
 52030.92096653 52361.64877196 52386.40388546 52390.297406
 52465.45370769 52546.31100788 52372.88632808 52026.91253138
 53939.59443937 53399.91899769 52573.81320173 52984.55582222
 53093.7162395  52965.51997819 52506.65463677 52983.39196071
 52646.75258768 52593.81747406 52553.17880709 52682.45857171
 52530.50461813 52705.67345016 53299.84543993 52023.68241077
 52856.566941   52436.95000056 52134.06138097 52664.18183273
 53050.93326468 52368.44994288 52734.8779888  52660.41786257
 51620.64617807 51891.87686314 52099.65869866 53079.33086235
 53312.95428706 52642.9594782  52473.16155747 52649.27767881
 52030.33493968 52676.81359481 52453.87872322 52563.58618171
 52634.74140432 53215.87806341 52927.4731986  52968.28751974
 52492.49310453 52370.19180481 52684.01432738 52455.60438761
 52637.79260748 52330.81859576 52464.36157386 52447.8635479
 52781.89994186 52720.18817657 52727.01997477 52732.82848232
```



```
 52935.88609964 53002.89868773 52815.60758473 52475.02332726
 52997.13636692 53100.44348013 51967.56425879 52989.69627439
 51876.47508408 52454.5490133  51887.29642249 52499.72864033
 52882.01082412 52867.16070085 52312.68035496 52448.83526015
 52880.17700668 52784.96379325 53196.24115793 52285.77719419
 53934.81947789 52185.99545914 51845.32281202 52763.12266086
 53014.29130787 52657.15801845 53408.48766218 52595.48748042
 52315.2757966  53703.48499624 52979.55389411 51853.03903862
 52276.36800776 52145.77770941 52742.60720989 51814.59312154
 52041.4616263  52286.38414501 52820.4951631  52586.01802664
 52193.18774453 52498.60742632 52941.07708728 52074.62109603
 53869.98991548 52584.909285   52536.48360531 52812.70980605
 52759.15881856 53118.01707834 52080.46313907 52700.41309441
 52464.26052298 52052.43072632 52564.8283483  52424.17040061
 53257.72397235 52171.21126699 53064.21379205 52536.83081253]

Process finished with exit code 0
```


Number of time-series segments per cluster
0
50
100
150
200
250
300
350
400
450
Cluster
0
5
10
15
20
25
30
35
40
45
50


Euclidean Distance between Closest Pair Segments in Cluster
43.952
41.952
39.952
1
2
3
4
5
6
Cluster



Execution Results Discussion

Based on the results, it seems that the data yields better results with more clusters with data when there are more clusters allocated through the k value. When the k value was allocated as 100, there were 11 non-empty clusters, but when the k value was allocated as 50, there were only 6 non-empty clusters. To further optimize this approach, dividing the data into the clusters more evenly could be leveraged through machine learning libraries. For the closest pair within each cluster, both k values yielded similar results with little variety in the distance between closest pairs across the different clusters. This shows that the data is consistent across the entire 1000 segments. In Kadane's algorithm, there was no difference in both k value executions since Kadane's algorithm relies on the specific time-series segments, rather than their clustering.

Conclusion

Throughout this project, the results shown were useful and understandable. Though the analysis of this project onto the PulseDB dataset, users can understand pulse data on a large scale, what an outlier pulse may look like, how to read time-series data, and more. In the medical field, it is crucial to properly analyze and process data to make future research accurate and reliable. In future works, it would be interesting to apply a time-series segment analysis on different medical data such as temperatures associations with illness, heartrates, and more to show how rates would cluster and the similarities between segments.

References

2.3. clustering. scikit. (n.d.). https://scikit-learn.org/stable/modules/clustering.html

alka1974. (2025, May 1). *Dynamic time warping (DTW) in time series*. GeeksforGeeks. https://www.geeksforgeeks.org/machine-learning/dynamic-time-warping-dtw-in-time-series/

Cecilia. (2019, July 18). *How to compare each element in array and calculate percentage of how many are the same in python?*. Stack Overflow. https://stackoverflow.com/questions/57098952/how-to-compare-each-element-in-array-and-calculate-percentage-of-how-many-are-th

Charya, S. (2025a, July 23). *Similarity search for time-series data*. GeeksforGeeks. https://www.geeksforgeeks.org/machine-learning/similarity-search-for-time-series-data/

clstaudt. (2018, August 6). *Comparing two numpy arrays for equality, element-wise*. Stack Overflow. https://stackoverflow.com/questions/10580676/comparing-two-numpy-arrays-for-equality-element-wise

CMPSC 463 Lecture Notes + Google Colab Notebooks

dadimadhav. (2025, July 15). *Calculate the euclidean distance using NumPy*. GeeksforGeeks. https://www.geeksforgeeks.org/python/calculate-the-euclidean-distance-using-numpy/

Giorgino, T. (2019). *DTW*. dtw - The dtw-python package 1.5.1 documentation. https://dynamictimewarping.github.io/py-api/html/api/dtw.dtw.html

Hierarchical clustering. Hierarchical clustering (scipy.cluster.hierarchy) - SciPy v1.16.2 Manual. (n.d.). https://docs.scipy.org/doc/scipy/reference/cluster.hierarchy.html

Kartik. (2025a, July 22). *Maximum subarray sum - Kadane's algorithm*. GeeksforGeeks. https://www.geeksforgeeks.org/dsa/largest-sum-contiguous-subarray/

Kartik. (2025b, July 23). *Minimum distance between two points*. GeeksforGeeks. https://www.geeksforgeeks.org/dsa/closest-pair-of-points-using-divide-and-conquer-algorithm/

Kartik. (2025c, August 22). *K means clustering – introduction*. GeeksforGeeks. https://www.geeksforgeeks.org/machine-learning/k-means-clustering-introduction/

larry. (2017, February 4). *How to plot pairwise distances of two-dimensional vectors?*. Stack Overflow. https://stackoverflow.com/questions/42041123/how-to-plot-pairwise-distances-of-two-dimensional-vectors

LeCheminant, R. (2020, August 31). *K means without libraries-python*. Medium. https://medium.com/data-science/k-means-without-libraries-python-feb3572e2eef

Matplotlib.pyplot. matplotlib.pyplot - Matplotlib 3.5.3 documentation. (n.d.). https://matplotlib.org/3.5.3/api/_as_gen/matplotlib.pyplot.html

Matplotlib 3.10.7 documentation. Matplotlib documentation - Matplotlib 3.10.7 documentation. (n.d.). https://matplotlib.org/stable/index.html

McDonald, A. (2024, December 16). *K-means clustering - an introduction*. Towards Data Science. https://towardsdatascience.com/k-means-clustering-an-introduction-9825ea998d1e/

Mishra, A. (2020, December 11). *Time series similarity using dynamic time warping -explained*. Medium. https://medium.com/walmartglobaltech/time-series-similarity-using-dynamic-time-warping-explained-9d09119e48ec

Modes, W. (2014, March 28). *Efficiently calculating a euclidean distance matrix using Numpy*. Stack Overflow. https://stackoverflow.com/questions/22720864/efficiently-calculating-a-euclidean-distance-matrix-using-numpy

NumPy documentation. NumPy. (n.d.). https://numpy.org/doc/

Priy, S. (2025, September 8). *Clustering in machine learning*. GeeksforGeeks. https://www.geeksforgeeks.org/machine-learning/clustering-in-machine-learning/

Quaink, B. (2021, November 17). *Vertically draw plot with matplotlib where each row in an array is a line*. Stack Overflow. https://stackoverflow.com/questions/69977702/vertically-draw-plot-with-matplotlib-where-each-row-in-an-array-is-a-line

Quick Start Guide. Quick Start Guide - h5py 3.15.1 documentation. (2014). https://docs.h5py.org/en/stable/quick.html#quick

rs736tjxi. (2025, September 30). *Time series clustering: Techniques and applications*. GeeksforGeeks. https://www.geeksforgeeks.org/machine-learning/time-series-clustering-techniques-and-applications/

Sayantan, J. (2025, April 15). *Numpy.squeeze() in Python*. GeeksforGeeks. https://www.geeksforgeeks.org/python/numpy-squeeze-in-python/

Sohail, S. (2018, March 16). *A comprehensive introduction to clustering methods*. Medium. https://shairozsohail.medium.com/a-comprehensive-introduction-to-clustering-methods-1e1e4f95b501

Wang, W. (2022, December 8). *PulseDB supplementary subsets*. Kaggle. https://www.kaggle.com/datasets/weinanwangrutgers/pulsedb-balanced-training-and-testing

youssef19. (2023, February 25). *Time series data visualization in Python*. Kaggle. https://www.kaggle.com/code/youssef19/time-series-data-visualization-in-python