Novel Method for Applying Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) in Computer Vision Analysis

Nikhil Shanbhag, Jason Ye, Kei Ogawa, Daniel Shi

May 2025

1 Problem Formulation

1.1 Overview

In computer vision, ascertaining the dimensions of objects and shapes within an image is an essential component of many endpoints in research and industry. Being able to calculate the dimensions of objects and shapes in images in a 2-dimensional plane also allows for application to video-based computer vision, where changes in detected object dimension can be related to changes in location within a 3-dimensional space. In this project, we aimed to develop an unsupervised method to estimate object dimensions. Starting from first principles, we analyzed binary images of simulated black shapes on a white background, and later we progressed our analysis to focus on a true use case for our method that applies a form of agglomerative clustering, named Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN).

Although clustering exists as an understood method in computer vision, our utilization of HDBSCAN is entirely novel in application and allows for robust estimation of object dimensionality in image with noise, essentially images with shapes that carry blurred features. This task is significant because blurred images can be challenging for object detection algorithms, especially in real-world applications like medical imaging [1], where accurate object measurement is critical. For example, in mammography [2] [3], the ability to detect anomalies or accurately estimate tumor size in blurred images can directly impact diagnosis and treatment decisions. Our project proposes utilizing HDBSCAN, an existing agglomerative clustering algorithm, to tackle this problem. By clustering the blurred regions in the images, HDBSCAN can help identify patterns and estimate dimensions of the objects within the image. The use of clustering is

especially beneficial since blurred regions tend to have similar pixel characteristics, and clustering can group these regions effectively while separating relevant pixel values from the background of the image, interpreted as noise.

1.2 Literature Review

"An Intuitionistic Fuzzy Clustering Approach for Detection of Abnormal Regions in Mammogram Images" [2] discusses the use of fuzzy clustering to detect abnormal regions in mammogram images, which is directly related to our goal of identifying and estimating object dimensions in blurred images. In medical images, such as mammograms, accurate detection of abnormal regions is critical for early diagnosis. The key takeaway from this literature would be the use of image processing and clustering algorithms to detect and quantify features (like abnormal regions or blurred objects) in medical images).

In addition, "An Outlier Removal Method Based on PCA-DBSCAN for Blood-SERS Data Analysis" [4] proposes an outlier removal method based on PCA-DBSCAN for blood-SERS analysis, focusing particularly on outliers. While this topic is not directly about imaging, the combination of PCA and DBSCAN is relevant to our use of HDBSCAN. The idea of outlier removal is essential in our blurry image detection task, as we may encounter noise or irrelevant data points (especially in blurry or low-quality images).

Lastly, "Moving Object Detection Based on Clustering and Event-Based Camera" [5] discusses moving object detection using clustering and event-based cameras, which shares a similar approach of using HDBSCAN to identify objects in images. For medical imaging and quality control applications, detecting objects within changing environments or under blurred conditions is similar to detecting moving objects in video or camera-based data. The clustering methods outlined in this paper could provide insights on how to handle spatial-temporal clustering, which might be beneficial for future reference if we extend our project to time-series data or video frames in medical imaging or quality control contexts.

2 Dataset Description

The dataset consists of entirely simulated data generated through a digital rendering software, Adobe Illustrator. The first binary images of a "near perfect" circle and a black rectangle on a white background were initially used for exploratory purposes in assessing the constraints of the HDBSCAN algorithm based on mathematical principles. We term the circle to be "near perfect" due to our realization that the graphical rendering software has a margin of error of ± 8 pixels when generating what it determines to be a standard circle.

Images are composed of pixels, which are a tuple of three integers, corresponding to three color channels: red (R), green (G), and blue (B). These values range

in value from 0 to 255, with values closer to 0 corresponding to a darker shade of the respective color channel. Thus, individual images can be thought of as a 3 dimensional collection of tensors, and each pixel can be thought of as being composed of individual vectors that correspond in value to the intensity of color for each color channel.

2.1 Deconvolution of Images

The images are deconvoluted from their tupled values into three separate 2-dimensional matrices representing the Red (R), Green (G), and Blue (B) color channels. Each matrix contains the pixel values corresponding to its respective color channel, which allows us to convert the image data into numerical form, where each column can be treated as an independent data vector. This process of deconvolution extracts and isolates the color information, transforming the original image into three distinct data matrices. These matrices are then saved in CSV format (R.csv, G.csv, B.csv), where each CSV file represents the spatial distribution of pixel intensities for respective color channels. Storing the data in CSV format allows for easy manipulation and serves as a structured input for subsequent analysis, such as clustering.

2.2 Test Cases: Binary Images

Test case 1 (Figure 1) is a white background with a black circle, although the circle is "near perfect", meaning the height and width differ slightly. The true dimension is 193 (H) $\times$ 201 (W). Test case 2 (Figure 2) is a white background with a black rectangle. The true dimension is 150 (H) $\times$ 300 (W). The true dimensions of these shapes were intuitively calculated through calculating the longest horizontal sequence of pixel values corresponding to black, which represents our width (W) and the longest vertical sequence (H). Because these images are binary, the only values in the deconvoluted matrices are 0 and 255, with 0 corresponding to the color, black, and 255 corresponding to the values of the white background.

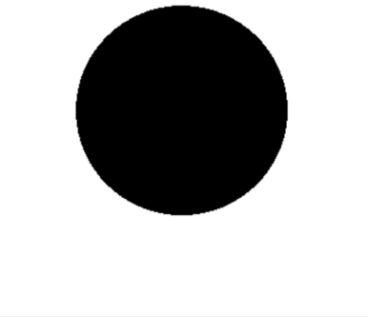


Figure 1: White Background with a Black Circle

Figure 2: White Background with a Black Rectangle

2.3 Test Cases: Blurred Images

The simulated dataset includes two blurred semi-circles and one blurred oval, providing a variety of shapes for testing the algorithm's ability to handle different object types. Blurred semi-circle 1 (Figure 3) below has true dimensions 118 (H) $\times$ 109 (W). Blurred semi-circle 2 (Figure 4) below has true dimensions 82 (H) $\times$ 91 (W). The final figure below, a blurred oval (Figure 5), has true dimensions 111 (H) $\times$ 179 (W). Due to the images being composed of a gradient of shades, the values for the R,G, and B color channels range from 0 to 255 continuously.

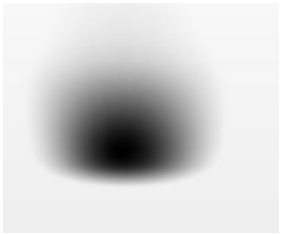


Figure 3: Blurred Semi-Circle 1

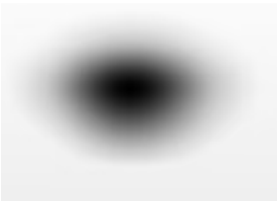


Figure 4: Blurred Semi-Circle 2

Figure 5: Blurred Oval

2.4 Simulated Data Example and Interpretation

To evaluate the structure of the HDBSCAN algorithm from first principles, we generated a synthetic dataset consisting of randomly distributed data points grouped into a specified number of clusters, while deliberately introducing outliers and noise (Figure 6). This synthetic dataset can be adapted to suit a variety of purposes, including the classification of malignant dermatological spots.

```
# Generate synthetic 2D data with 6 clusters
X, _ = make_blobs(n_samples=300, centers=6, n_features=2, cluster_std=0.5,
                  random_state=42)

# Add some random noise/outliers
outliers = np.random.uniform(low=-10, high=10, size=(20, 2))
X = np.vstack([X, outliers])
```

Figure 6: Code Snippet to Generate Synthetic Data with Outliers and Noise

After applying HDBSCAN, the algorithm successfully identified and separated the clusters, as shown in the projection onto component X and component Y (Figure 7). Notably, the outliers were distinctly classified as noise, demonstrating the algorithm's capability to handle data irregularities while preserving the composition of the main clusters.

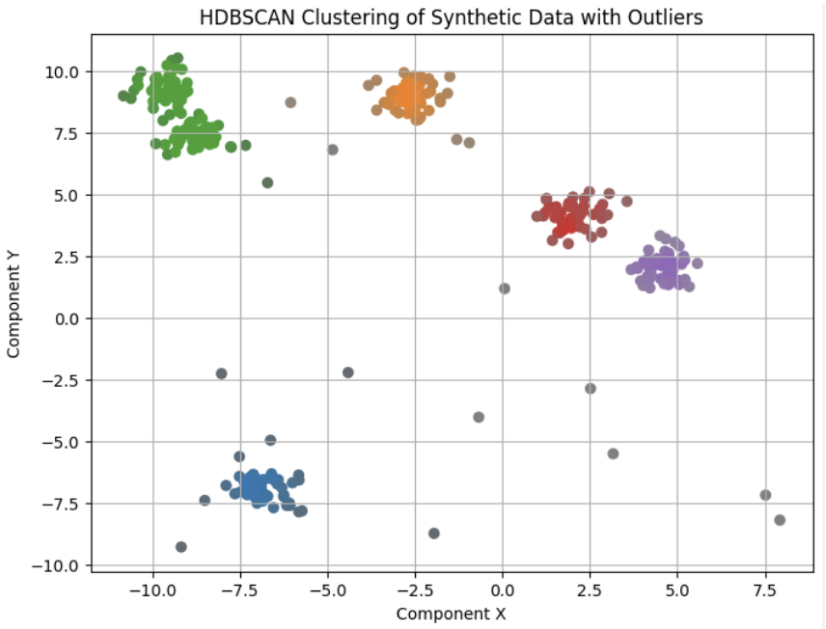


Figure 7: HDBSCAN Clustering of Synthetic Data with Outliers and Noise

The condensed tree in HDBSCAN (Figure 8) provides a hierarchical visualiza-

tion of the clustering structure and helps assess the stability of detected clusters. In this plot, the vertical axis represents the cluster's persistence across varying density thresholds, with higher branches indicating more stable and significant clusters. Each branch corresponds to a group of points, and splits show how clusters divide as the density threshold increases. Stable clusters are typically highlighted in color, while less stable or insignificant clusters appear as gray branches. The width and height of a branch reflect the relative strength and size of the cluster, with flatter, longer branches indicating well-separated, robust clusters. Outliers and noise are often short, isolated branches or are filtered out entirely, allowing us to differentiate between meaningful clusters and noise.

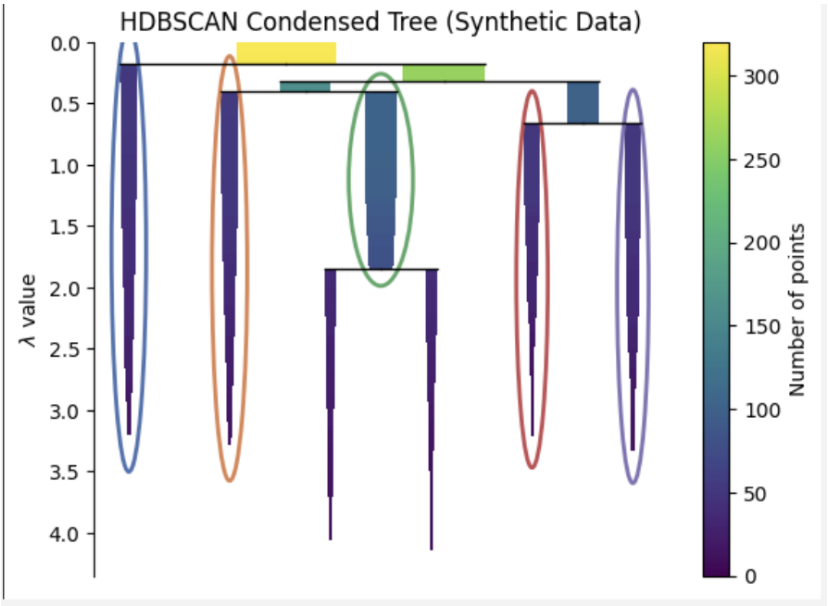


Figure 8: HDBSCAN Condensed Tree of Synthetic Data with Outliers and Noise

3 Methodology and Analysis

3.1 Motivation Behind HDBSCAN

HDBSCAN is a density-based clustering algorithm that identifies clusters of varying densities and shapes using distance metrics. Unlike methods like K-Means, which assume spherical clusters and require predefining the number of clusters, HDBSCAN automatically determines both the number and structure of clusters based on data distribution. Studies have shown HDBSCAN consistently outperforms K-Means and similar algorithms, particularly in image segmentation tasks involving complex gradients, irregular patterns, and noise [6].

Our initial consideration was that algorithms such as K-Means, though computationally efficient, would struggle to detect complex gradients in pixel values, such as abrupt changes in color or hues along curves. Therefore, we decided to use HDBSCAN as it is meant to resolve exactly this issue. This method is particularly useful for data with non-uniform densities or complex shapes. It builds a hierarchy of density-based clusters, allowing multi-resolution analysis, and can automatically detect noise and outliers. Additionally, it supports soft clustering with membership probabilities and is robust to varying densities, making it well-suited for noisy or imprecise data.

3.2 Mathematics Behind HDBSCAN

3.2.1 Step 1: Mutual Reachability Distance

Let $d(a, b)$ denote the standard Euclidean distance between points a and b. The *core distance* [7] of a point x, denoted as $\text{core}_k(x)$, is defined as the distance from x to its k-th nearest neighbor, where k is a user-defined parameter often referred to as min_samples. Formally, this can be expressed as:

$$\text{core}_k(x) = d(x, \text{k-th nearest neighbor of } x).$$

Using the concept of core distance, we define the *mutual reachability distance* between two points x and y as:

$$d_{\text{mreach}}(x, y) = \max\left\{\text{core}_k(x), \text{core}_k(y), d(x, y)\right\}.$$

The intuition behind the mutual reachability distance is to "stretch" sparse regions of the data space while allowing dense regions to remain compact. By taking the maximum of the core distances of x and y and their actual Euclidean distance, this metric effectively increases the distance between points in sparse areas, thus solving the problem of varying densities when clustering.

3.2.2 Step 2: Minimal Spanning Tree Construction

A *mutually reachable graph* [8] is defined as a graph in which there exists a path from one node to another and vice versa, ensuring bidirectional connectivity between nodes. To construct such a graph, we first build a complete weighted graph $G = (V, E)$, where:

- $V = X$, meaning each data point corresponds to a vertex in the graph.
- Each edge (x_i, x_j) is assigned a weight $w_{ij} = d_{\text{mreach}}(x_i, x_j)$, where d_{mreach} denotes the mutual reachability distance.

A *spanning tree* T of G is a connected, acyclic subgraph that includes all vertices in V. Among all possible spanning trees, we are interested in constructing the *minimum spanning tree* (MST), which minimizes the total sum of edge weights. Formally, the MST can be defined as:

$$T^* = \underset{T \text{ spanning } G}{\arg\min} \sum_{(x_i, x_j) \in T} d_{\text{mreach}}(x_i, x_j).$$

3.2.3 Step 3: Hierarchical Clustering via Single Linkage

The minimum spanning tree (MST) is utilized to perform *hierarchical agglomerative clustering* using the *single linkage* criterion [9]. Clusters are formed by iteratively merging the closest connected components as follows:

- Sort the edges of the MST T by their weights: $e_1, e_2, \ldots, e_{n-1}$.
- For each edge $e_k = (x_i, x_j)$ at merge step k, update the clustering as:

$$C(k) = C(k-1) \setminus \{C_i, C_j\} \cup \{C_i \cup C_j\},$$

  where C_i and C_j are the clusters containing points x_i and x_j, respectively, merged at distance $k = d_{\text{mreach}}(x_i, x_j)$.
- The sequence $\{k\}_{k=1}^{n-1}$ defines the merge heights of the dendrogram.
- Final clusters are extracted by optimizing a *stability* criterion (discussed in Step 5) over the resulting hierarchy.

3.2.4 Step 4: Cluster Hierarchy Extraction

The result of this process is a *dendrogram*, which is a tree structure where each level corresponds to a specific density threshold. For a dataset X with minimum spanning tree T and edges $e_1, \ldots, e_{n-1}$, the dendrogram is defined by the set of connected components:

$$C = \text{connected components of } \{e_k \mid w_k\}, \quad \text{for} \quad \epsilon \in [0, \infty).$$

As such, finer partitions are obtained for lower values of ϵ. The dendrogram's structure is directly determined by the MST edge weights, where each merge occurs at $\epsilon_k = w_k = d_{\text{mreach}}(x_{i_k}, x_{j_k})$.

Unlike DBSCAN, which uses a fixed density threshold ϵ, HDBSCAN considers all possible density levels. DBSCAN creates a single flat clustering [10]:

$$C_{\text{DBSCAN}} = C(\epsilon),$$

which corresponds to a single partition at the chosen ϵ. In contrast, HDBSCAN builds a hierarchy of partitions:

$$\{C(\epsilon) \mid \epsilon \in [0, \infty)\},$$

where small values of ϵ correspond to finer partitions, and clusters in high-density regions are merged as ϵ increases.

3.2.5 Step 5: Condensed Tree and Stability Calculation

A *condensed tree* is built by simplifying the dendrogram, removing unnecessary branches that do not contribute to meaningful clusters. The *stability* [11] of a cluster C is computed using the formula:

$$\text{stability}(C) = \sum_{x \in C} \left(\lambda_{\text{death}}(x) - \lambda_{\text{birth}}(x) \right),$$

where:

- $\lambda_{\text{death}}(x)$ is the density level at which point x "falls out" of the cluster,
- $\lambda_{\text{birth}}(x)$ is the density level at which point x "joins" the cluster.

3.2.6 Step 6: Optimal Flat Clustering Extraction

The most persistent [12] clusters, defined as those exhibiting high stability across multiple density levels, are selected by iteratively traversing the condensed tree and choosing clusters that maximize stability. A cluster C_v is considered optimal if it satisfies the condition:

$$\text{stability}(C_v) + \sum_{\text{grandchildren } u} S_{\text{opt}}(u) > \sum_{\text{children } w} S_{\text{opt}}(w),$$

where $S_{\text{opt}}(\cdot)$ denotes the optimal stability of a cluster. The total stability of the clustering hierarchy is given by:

$$\text{Total Stability} = S_{\text{opt}}(\text{root}).$$

3.3 Novel Process for Detecting Image Dimensions and Discovery

When it came to answering our initial research question, where we opted to explore how the concept of clustering itself could be utilized in computer vision, we were constrained to investigate the topic through first principles. Intuitively, thinking of how we could elucidate data from the binary images, it became clear that the boundaries of the pixel values 0 and 255 told a story of the true dimensions of a shape. Thinking of the columns of the deconvoluted matrices of the image as independent vectors, it's clear that general clustering, such as K-means, should detect values of 0, nested between values of 255, corresponding to the white background, as singular clusters. This would point towards detection of our shape within the background, however, it would not necessarily yield useful information about the dimensions of the shape.

To build upon this knowledge gap, we discovered the existence of HDBSCAN, which showed promise in allowing us to explore image dimensionality. While simple clustering methods can focus on individual columns treated as an array of

integers, agglomerative clustering techniques, such as HDBSCAN, would allow us to focus on the relationship of independent $(n \times 1)$ columns that construct the image together. When an $(n \times p)$ matrix is inputted into HDBSCAN, it operates on columns of the matrix in a left to right sequence for p iterations, calculating mutual reachability distance, stability, and the existence of core clusters, compiling these values in generating a hierarchical linkage between said $(n \times 1)$ columns.

The most convenient parameter to tune in HDBSCAN is the `min_cluster_size` parameter, which directly involves the sensitivity of clustering in determining the minimum number of clustered values that are required to be considered a core cluster (specified in section 3.2.1). In our initial testing of binary images for example, specifying a value of 2 for this parameter would allow two adjacent values of 0, corresponding to two black pixels in the image, to be considered a cluster. In another example, specifying a value bigger than the longest continuous sequence of 0's, such as 500, would result in 0 clusters being detected, due to the nonexistence of such a large Euclidean distance of points within the image itself, which would result in an empty tree diagram (Figure 11).

Understanding the fundamentals of the mathematical underlying principles of HDBSCAN allowed us to motivate and discover a true novel use case for applying clustering towards the computer vision endpoint of ascertaining dimensionality. *Notably, continuously testing values of min cluster size values iteratively in ascending values until 0 clusters are detected should yield actionable information on the maximal amount of adjacent values within our image corresponding to one dimension our shape or object of interest.* We term this later as our novel "sweep until noise" process.

Operating the above heuristic on an $(n \times p)$ matrix obtained from one of the RGB channels of an image should directly relate to the maximum height of a shape within an image, as HDBSCAN operates on the input matrix from left to right, so the data retrieved from clustering is resultant from it acting on the vertical axis of the actual image. Intuitively, computing the transpose matrix of the original matrix would allow the same process to be applied to yield information on the horizontal axis. Therefore, the algorithm acting on the vertical axis can be said to correspond to the calculation of shape height (H), and the algorithm acting on the horizontal axis can be said to correspond to the shape width (W).

It is also important to note that in the case of our testing on grayscale images, due to the only color being black, gray, or white, the R,G, and B color channel matrices are essentially equivalent. For that reason, performing clustering on an individual channel per image is equivalent to repeating the process for the other color channels.

The original exploratory test case applies HDBSCAN to analyze the dimensions of binary images. Specifically, a black circle on a white background is used to

calculate its diameter, and a black rectangle is used to determine its height and width. The use of HDBSCAN is then extended onto two blurred semi-circles and one blurred oval, demonstrating clustering performance on imprecise shapes.

Our approach (Figure 9) iterates over subplots representing width and height. It initializes empty lists to track `min_cluster_size` and cluster counts, starting from 5. For each iteration, HDBSCAN is fitted to the data or its transpose. The `min_cluster_size` value at which the number of clusters drops to zero is used to infer the height (from the original matrix) and width (from the transposed matrix) of the shapes.

```
FOR each subplot (Height and Width):
    Initialize empty lists to track min_cluster_size and number of clusters
    Set starting min_cluster_size = 5

    WHILE number of clusters > 0:
        Fit HDBSCAN clustering on data (or its transpose)
        Count number of non-noise clusters (labels ≠ -1)
        Store current min_cluster_size and cluster count

        IF cluster count == 0:
            Break the loop
        ELSE:
            Increment min_cluster_size by 1

    Plot min_cluster_size vs number of clusters
    Mark the point where no clusters are found
```

Figure 9: Pseudocode for Determining Image Dimensions

There are two main visualizations accompanying the analysis. The first is a condensed tree illustrating cluster membership information (Figure 10, 11). Figure 10 demonstrates an example of a tree diagram generated from black shape on a white background. Notably, only two clusters are detected, corresponding to the white background RGB values and the values corresponding the the black pixels of the shape. Figure 11 demonstrates a negative control where a blank image of only a white background (an input matrix of with only values of 255).

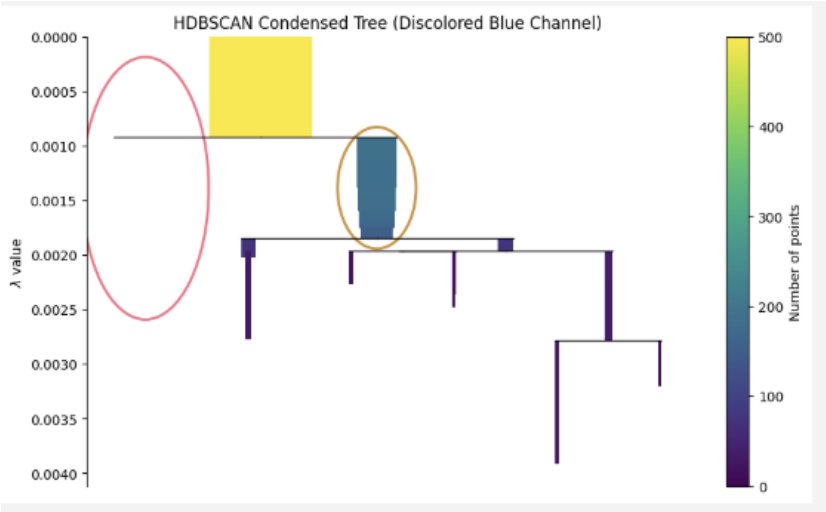


Figure 10: Example Condensed Tree for Discolored Blue Channel

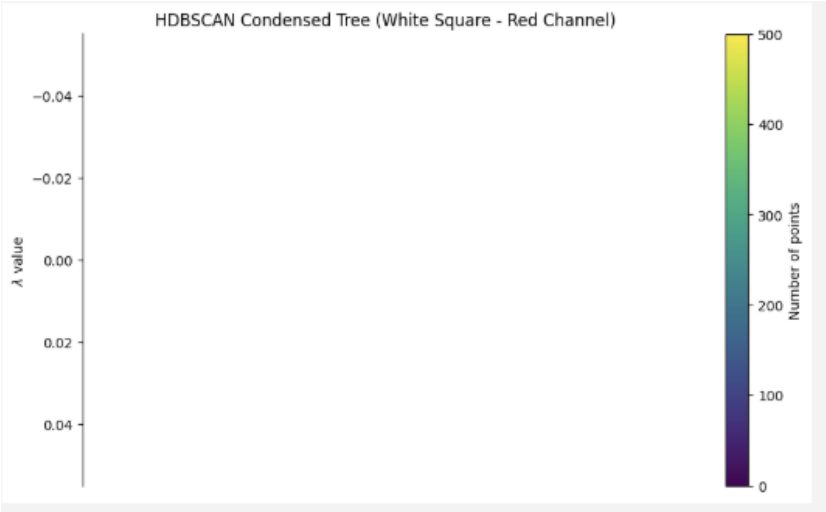


Figure 11: Example Condensed Tree for White Square and Red Channel (No Clustering)

The following figures demonstrate how the number of clusters varies with different values of `min_cluster_size` (Figure 12, 13). This plot exhibits the results of our novel process in iterating values of the parameter until failure to detect clusters. As seen in the figure, lower values of "min cluster size" correspond with more frequent characterization of core points, whereas raising the criteria for

clustering by increasing "min cluster size" values directly reduces the amount of clusters detected.

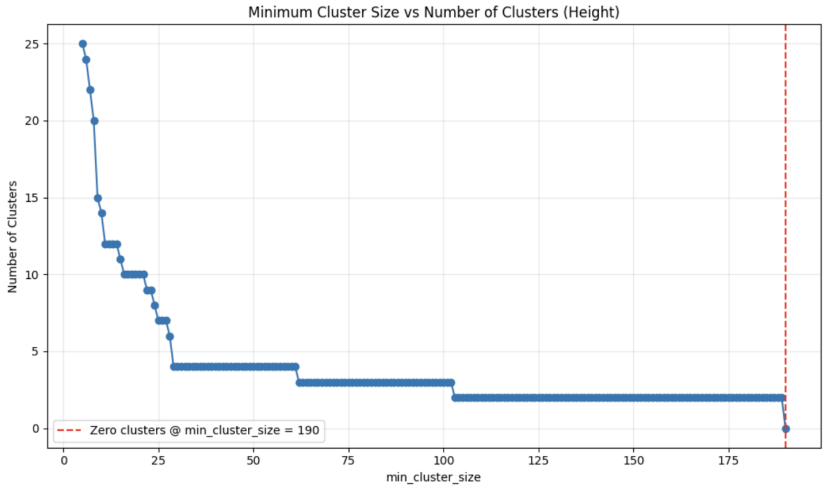


Figure 12: Example Output for Dimension Detection of Height

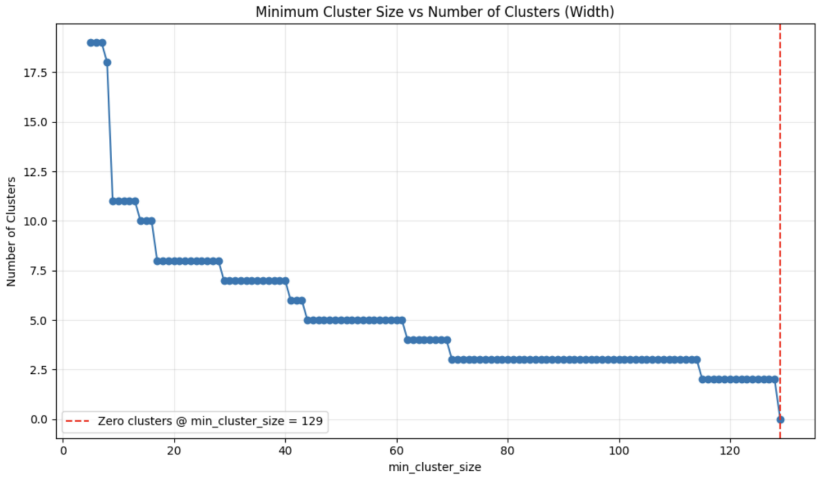


Figure 13: Example Output for Dimension Detection of Width

4 Results and Discoveries

The HDBSCAN algorithm was able to correctly determine the dimensions of the width and height of the near perfect circle (Figure 14, 15) and blurred images

(Figure 19, 20, 22, 23, 25, 26). However, it was not able to correctly determine the dimensions of the rectangle (Figure 16, 17). For our images containing blurred features, the height and width of the object were determined manually using a conventionally used image analysis software, ImageJ. Height and width for solid shapes were calculated directly and empirically through calculating the longest sequence of 0's within the resultant matrices of the color channels.

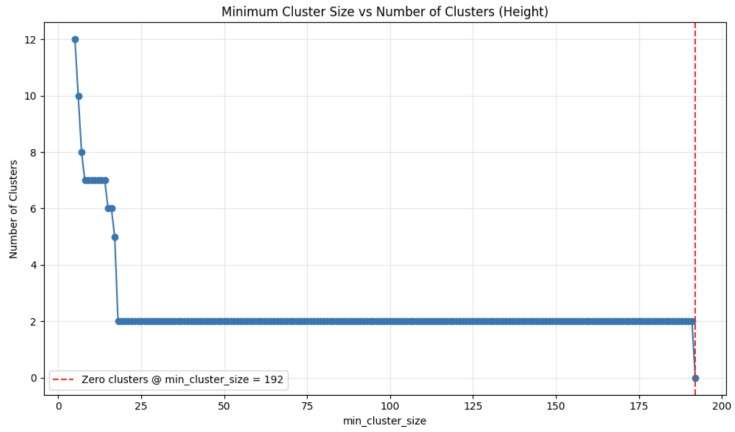


Figure 14: Height Measurement of Near Perfect Circle

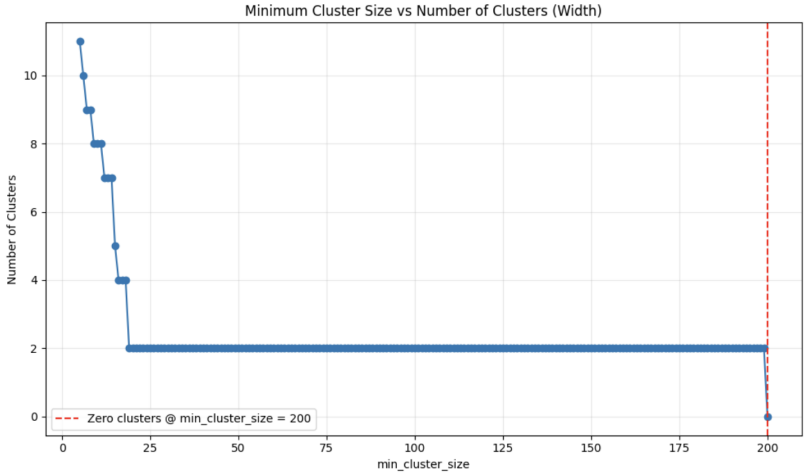


Figure 15: Width Measurement of Near Perfect Circle

Due to the non-smooth edges of a rectangle, the height of the rectangle (Figure 16) continuously clusters and then suddenly stops. However, the width (Figure

17) clearly ends clustering early, as it underestimating by approximately 100 pixels. This will be further explored in subsequent sections of (*Discussion*).

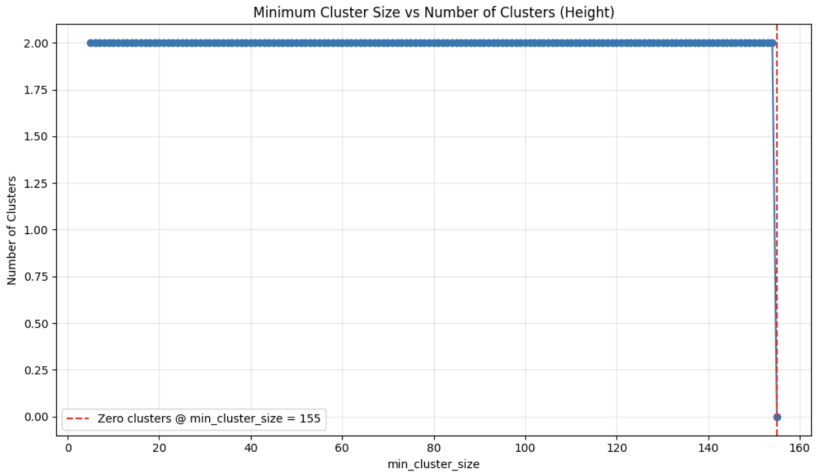


Figure 16: Height Measurement of Rectangle

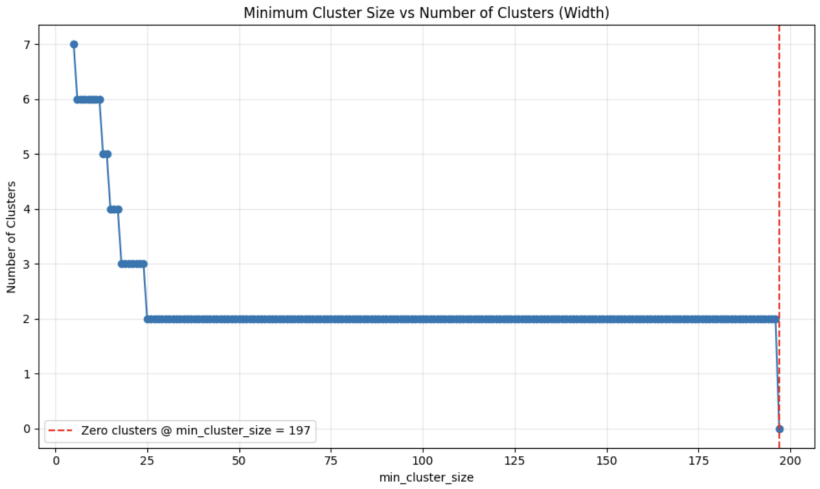


Figure 17: Width Measurement of Rectangle

In the condensed tree plot of Figure 18, 21, and 24, red circles indicate the clusters that the algorithm has selected as the most stable and meaningful based on their persistence across varying density thresholds. These clusters are chosen because they maintain their structure over a wide range of density levels, suggesting they are well-defined. The absence of red circles in certain branches

typically implies that those areas are considered noise or contain unstable clusters that were not selected.

As shown in Figure 19 and 20, the clustering is more effective and accurate for blurred semi-circle 1 when it comes to both the height and width.

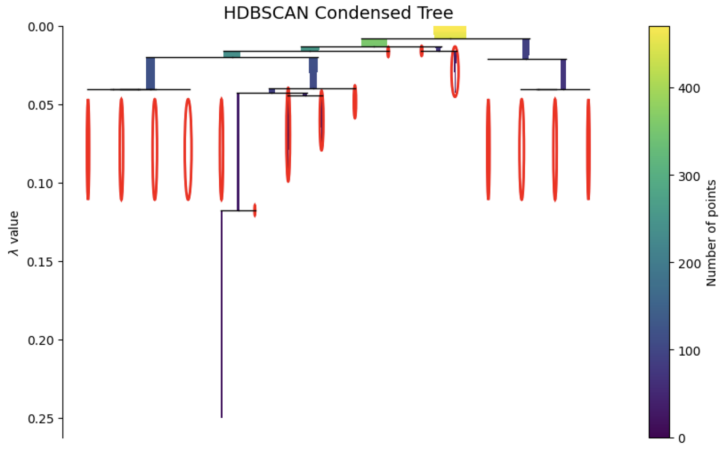


Figure 18: Condensed Tree of Blurred Semi-Circle 1

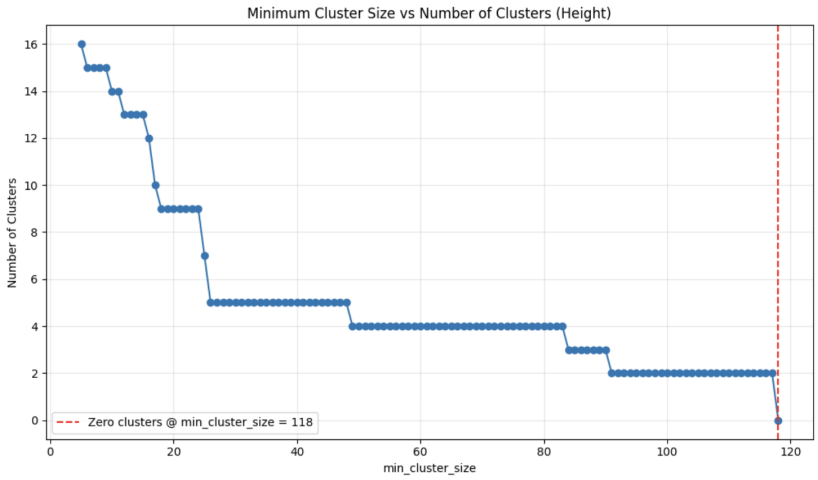


Figure 19: Height Measurement of Blurred Semi-Circle 1

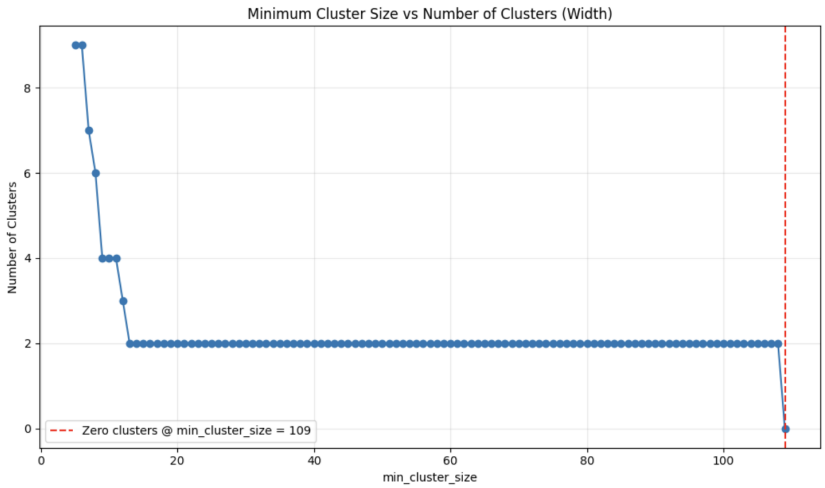


Figure 20: Width Measurement of Blurred Semi-Circle 1

The same type of effective clustering is shown below in Figure 22 and 23 for blurred semi-circle 2.

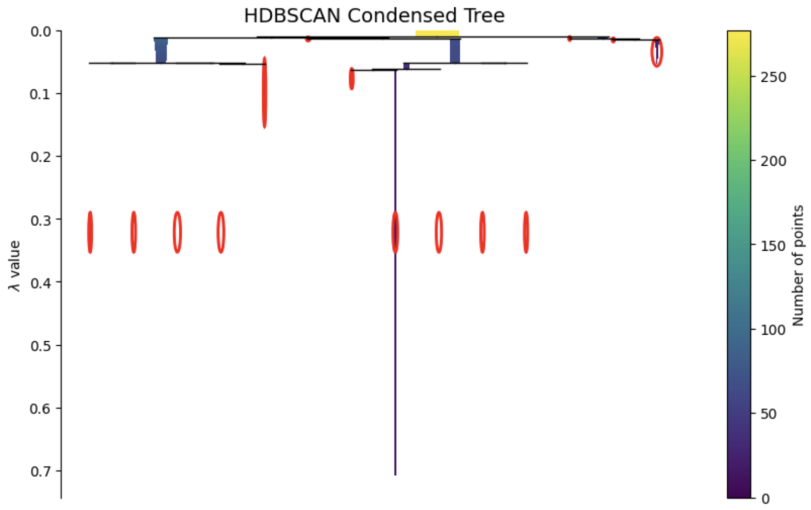


Figure 21: Condensed Tree of Blurred Semi-Circle 2

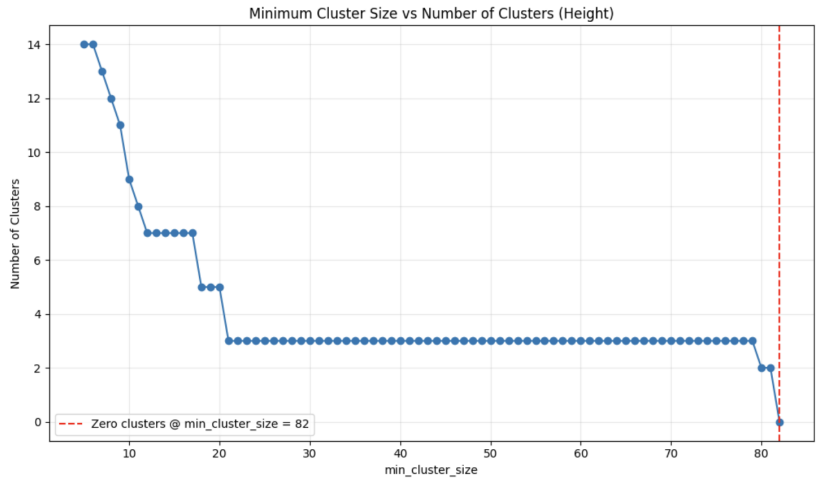


Figure 22: Height Measurement of Blurred Semi-Circle 2

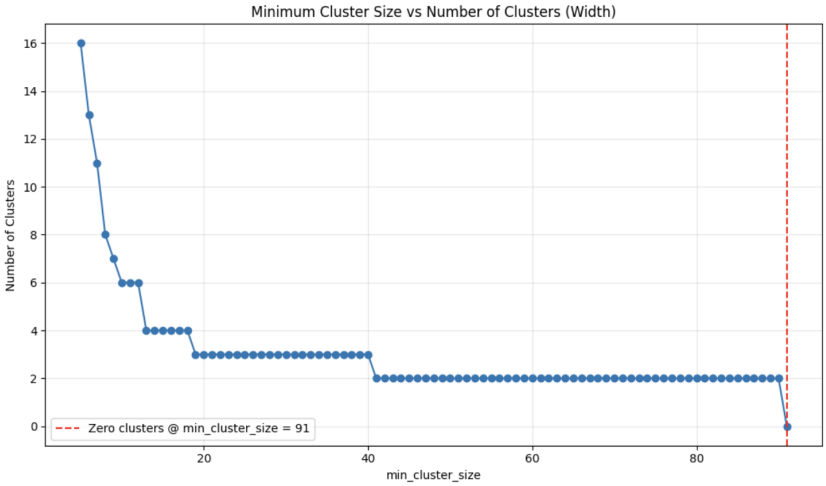


Figure 23: Width Measurement of Blurred Semi-Circle 2

Finally, we see the same trend for the blurred oval in Figure 25 and 26. Due to the longer curvature of the figure, the number of clusters drops off immediately as the minimum cluster size increases.

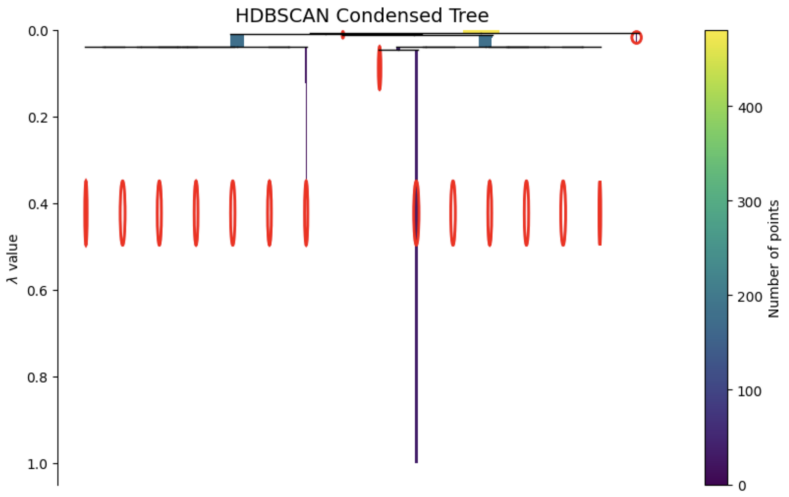


Figure 24: Condensed Tree of Blurred Oval

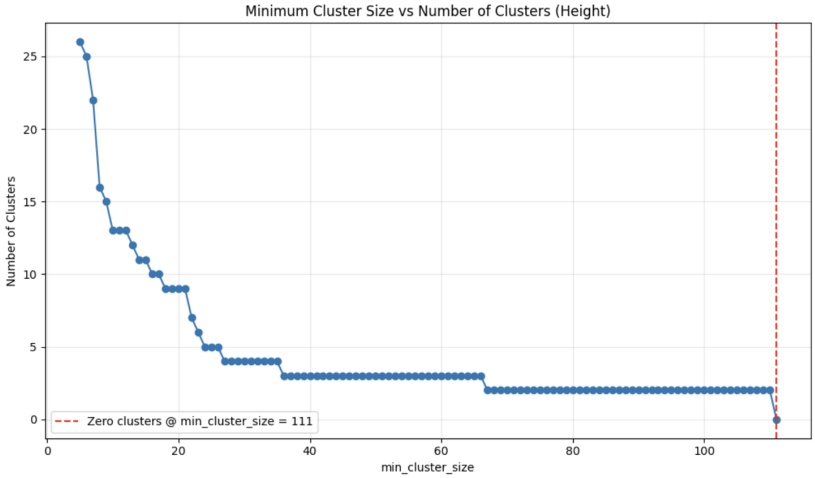


Figure 25: Height Measurement of Blurred Oval

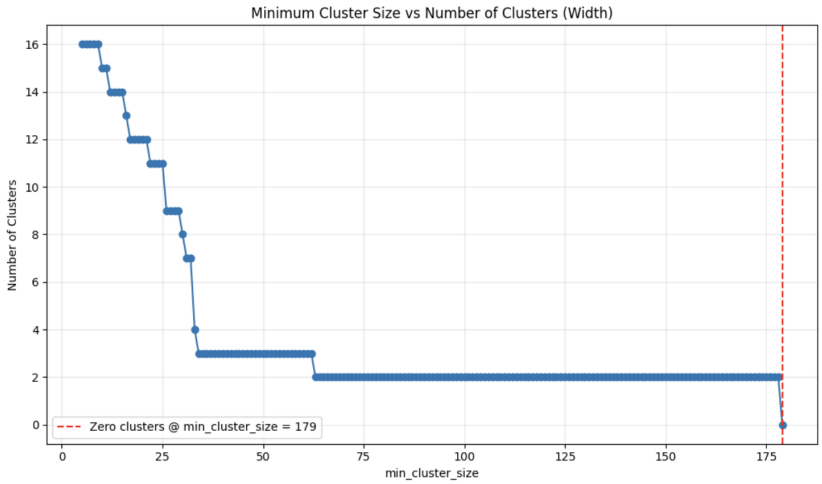


Figure 26: Width Measurement of Blurred Oval

HDBSCAN performs well on circles because their geometry ensures that each column of pixels is unique. Unlike rectangles, which have long flat edges resulting in identical column vectors, a circle's outline changes gradually from column to column. This variation creates meaningful distances between neighboring columns in HDBSCAN's graph, preventing premature merging of columns into a single cluster. As a result, the algorithm identifies a cohesive cluster whose width corresponds to the circle's true diameter.

Since the density of neighboring columns remains consistent across the circle, increasing `min_cluster_size` gradually prunes small structures while preserving the main cluster. The circle's cluster persists until the `min_cluster_size` value reaches the circle's actual width, at which point the cluster dissolves, revealing the correct diameter. In essence, the circle's curved boundary ensures distinct data representations that align well with HDBSCAN's clustering logic, making the diameter extraction accurate.

5 Discussion

5.1 Blurred Shapes

The most notable result from our investigation was the robustness of the calculation of dimensions for the blurred shapes included in our dataset. Our "sweep until noise" process essentially managed to separate the visually intuitive blurred shape's dimensions from what is visually intuitive as noise, the gray shading of the image. This highlights how our method is able to capture the visual logic of what constitutes the primary shape of a pictured object containing noise, in this case blurred shading. This is less than coincidental when considering the

mathematical processes behind clustering. With values closer to 0 representing black, and values closer to 255 representing the color white, clustering allows us to group the densely packed darker pixels from the light gray and white background due to these pixel color values being treated as individual vectors.

5.2 Limitations

The density-based clustering algorithm HDBSCAN struggles when applied to measure the width of binary silhouettes by encoding image columns as high-dimensional vectors. While it seems intuitive to represent each column of pixels as a binary vector and apply clustering to identify foreground regions, this method fails in the presence of shapes with long, flat sides, such as rectangles or elongated ovals. The root cause is a degeneracy in feature space, where identical columns collapse into a single point in HDBSCAN's internal graph representation.

For example, in a rectangle of height h, every interior column shares the same binary pattern, resulting in identical vectors with zero pairwise distances. Consequently, HDBSCAN merges these columns immediately, reducing a potentially wide object to just a few unique edge vectors. This premature collapse undermines the "sweep until noise" approach often used to estimate width, as the apparent cluster size becomes a significant underestimate of the true object width.

As `min_cluster_size` increases, the algorithm reaches the "all noise" state far earlier than expected, misinterpreting dense duplicates as ultra-stable micro-clusters while discarding the true geometric information. HDBSCAN's reliance on minimum-spanning trees and excess-of-mass hierarchies makes it suitable for continuous point clouds, not for data with exact duplicates as seen in silhouette encoding.

In contrast, HDBSCAN performs well on circles, where each column's height varies smoothly, preventing duplicate vectors and preserving meaningful geometric relationships. For shapes with flat edges, however, alternative methods like connected-component analysis or column-wise indexing are better suited. Without addressing the duplicate issue—either by injecting noise or rethinking the approach—HDBSCAN cannot reliably recover widths of such silhouettes.

5.3 Conclusion and Future Extensions

In summation, understanding the principles underlying clustering and rigorously testing the limitations of agglomerative clustering allowed us to approach discovering a new form of analysis in computer vision. While our sample of images used in testing are controlled and limited in features, further testing of different forms of images can be performed to unveil limitations and additional novel use cases for our method.

Future extensions of this project could involve enhancing the clustering methodology to handle motion blur, varying lighting conditions, or partial occlusions, thereby increasing its robustness in real-world scenarios. Additionally, the approach could be extended beyond simple geometric shapes to detect and analyze irregular or overlapping objects, which are common in medical and industrial imaging. Integrating deep learning-based feature extraction could further improve the accuracy of object detection and dimension estimation in blurred images. Ultimately, these extensions would broaden the applicability of the project to diverse fields like biomedical diagnostics, remote sensing, quality control in manufacturing, and automated surveillance systems.

References

[1] Chen, T.-J., Chuang, K.-S., Chang, J.-H., Shiao, Y.-H., & Chuang, C.-C. (2006). A blurring index for medical images. *Journal of Digital Imaging*, *19*(2), 118–125. https://doi.org/10.1007/s10278-005-8736-y

[2] Chaira, T. (2021). An intuitionistic fuzzy clustering approach for detection of abnormal regions in mammogram images. *Journal of Digital Imaging*, *34*(2), 428–439. https://doi.org/10.1007/s10278-021-00444-3

[3] Nowakowska, S., Vescoli, V., Schnitzler, T., Ruppert, C., Borkowski, K., Boss, A., Rossi, C., Wein, B., & Ciritsis, A. (2024). Technical feasibility of automated blur detection in digital mammography using convolutional neural network. *European Radiology Experimental*, *8*, Article 129. https://doi.org/10.1186/s41747-024-00527-0

[4] Liu, M., Wang, T., Zhang, Q., Pan, C., Liu, S., Chen, Y., Lin, D., & Feng, S. (2024). An outlier removal method based on PCA-DBSCAN for blood-SERS data analysis. *Analytical Methods*, *16*, 846–855. https://doi.org/10.1039/D3AY02037A

[5] H. Abu-Mariah and W. Ashour, *Moving Object Detection Based on Clustering and Event-Based Camera*, IEEE, 2023. Available: https://ieeexplore.ieee.org/document/10209469

[6] J. Wang, Y. Wang, J. Wang, Y. Shen, and X. Zhao, "Design of an Image Segmentation System Based on Hierarchical Density-Based Spatial Clustering of Applications with Noise for Off-Road Unmanned Ground Vehicles," in *Proceedings of the 2022 International Conference on Autonomous Unmanned Systems (ICAUS 2022)*, vol. 1010, Lecture Notes in Electrical Engineering, Springer, 2023, pp. 3163–3175. https://doi.org/10.1007/978-981-99-0479-2_291

[7] McInnes, L., Healy, J., & Astels, S. (2023). *How HDBSCAN Works*. Retrieved from https://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html

[8] S. Singh, *Depth-First Search and Directed Graphs*, CS256: Algorithm Design and Analysis, Williams College, Fall 2019. Available: https://www.cs.williams.edu/~shikha/teaching/fall19/cs256/lectures/Lecture04.pdf

[9] J. C. Gower and G. J. S. Ross, "Minimum Spanning Trees and Single Linkage Cluster Analysis," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 18, no. 1, pp. 54–64, 1969. https://www.jstor.org/stable/2346439

[10] McInnes, L., Healy, J., & Astels, S. (2023). *Extracting DBSCAN* clustering from HDBSCAN**. Retrieved from https://hdbscan.readthedocs.io/en/latest/dbscan_from_hdbscan.html

[11] LaViale, T. (2023). *Understanding HDBSCAN: A Deep Dive into Hierarchical Density-Based Clustering*. Arize AI. Retrieved from https://arize.com/blog-course/understanding-hdbscan-a-deep-dive-into-hierarchical-density-based-clustering/

[12] Rolle, A. (2018). *The HDBSCAN* Clustering Algorithm*. Presented at the Topological Data Analysis Seminar, University of Western Ontario. Retrieved from https://jdc.math.uwo.ca/TDA/Rolle-clustering.pdf

6 Code Appendix

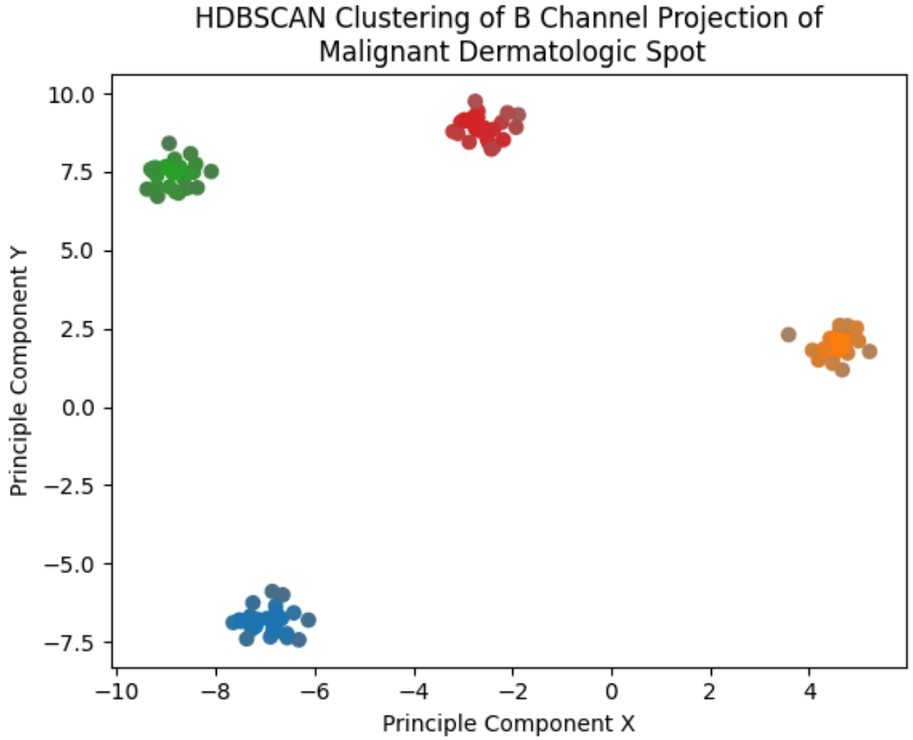
```
import numpy as np
from PIL import Image
import sys

def deconvolute_image_to_csv(input_image_path):
    """
    Reads a PNG image from input_image_path and deconvolutes it into three matrices (R, G, B).
    Saves each channel separately as R.csv, G.csv, and B.csv.
    """
    # Open image and ensure it's in RGB mode
    image = Image.open(input_image_path).convert('RGB')

    # Convert the image to a NumPy array
    img_array = np.array(image)
    # img_array has shape: (height, width, 3)

    # Separate the array into R, G, B
    R = img_array[:, :, 0]
    G = img_array[:, :, 1]
    B = img_array[:, :, 2]

    # Save each channel to CSV
    np.savetxt("R.csv", R, fmt='%d', delimiter=",")
    np.savetxt("G.csv", G, fmt='%d', delimiter=",")
    np.savetxt("B.csv", B, fmt='%d', delimiter=",")

    print("Successfully saved R.csv, G.csv, and B.csv")

if __name__ == "__main__":
    if len(sys.argv) < 2:
        print("Usage: python deconvolute_image.py <input_image_path>")
        sys.exit(1)

    input_path = sys.argv[1]
    deconvolute_image_to_csv(input_path)
```

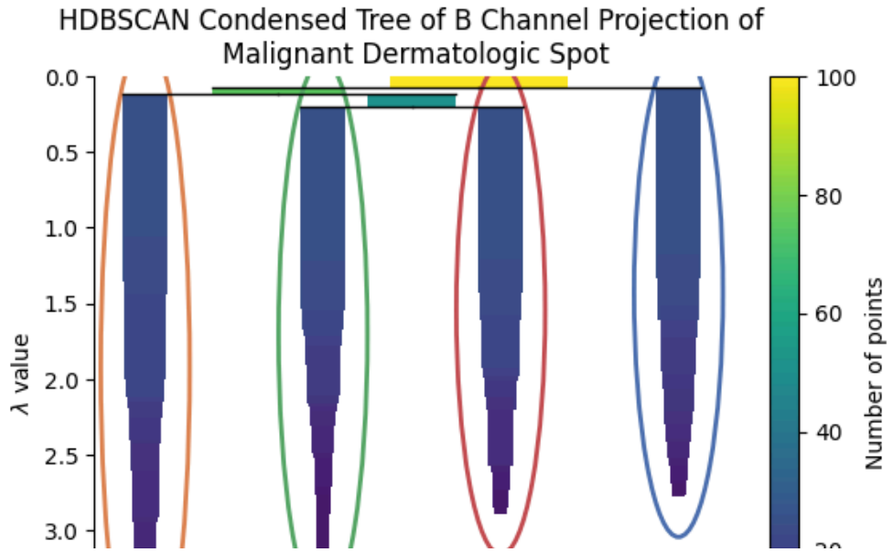
Randomized Matrix Results

```
import hdbscan
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import statistics
from itertools import chain
from sklearn.datasets import make_blobs

import warnings
warnings.filterwarnings("ignore")

# Generate synthetic 2D data with 4 clusters
X, _ = make_blobs(n_samples=100, centers=4, n_features=2, cluster_std=0.4,
                  random_state=42)

# Convert to DataFrame
df = pd.DataFrame(X, columns=["X", "Y"])

# Save to CSV
df.to_csv("synthetic_clusters.csv", index=False, header=False)

projection = np.loadtxt("synthetic_clusters.csv", delimiter=",")
#print(projection)


#/// converting to list
lis = projection.tolist()
print (lis)


#///-----------------------------------clustering algorithm-------------------------------------------------------
clusterer = hdbscan.HDBSCAN(min_cluster_size=6, gen_min_span_tree=True)
clusterer.fit(projection)


palette = sns.color_palette()
cluster_colors = [sns.desaturate(palette[col], sat)
                  if col >= 0 else (0.5, 0.5, 0.5) for col, sat in
```

```
                zip(clusterer.labels_, clusterer.probabilities_)]


fig = plt.scatter(projection.T[0], projection.T[1], c=cluster_colors)
plt.title('HDBSCAN Clustering of B Channel Projection of\n Malignant Dermatologic Spot')
plt.xlabel('Principle Component X')
plt.ylabel('Principle Component Y')
plt.show()


clusterer.condensed_tree_
clusterer.condensed_tree_.plot(select_clusters=True, selection_palette=sns.color_palette('deep',8))
plt.title('HDBSCAN Condensed Tree of B Channel Projection of\n Malignant Dermatologic Spot')
plt.show()
```

```
[[-9.177045243863372, 6.717584025624357], [-6.554220304203402, -7.372455319849529], [-2.6969873774267312, 9.231310145632708], [-6.31850946677683, -7.440850018392859
```

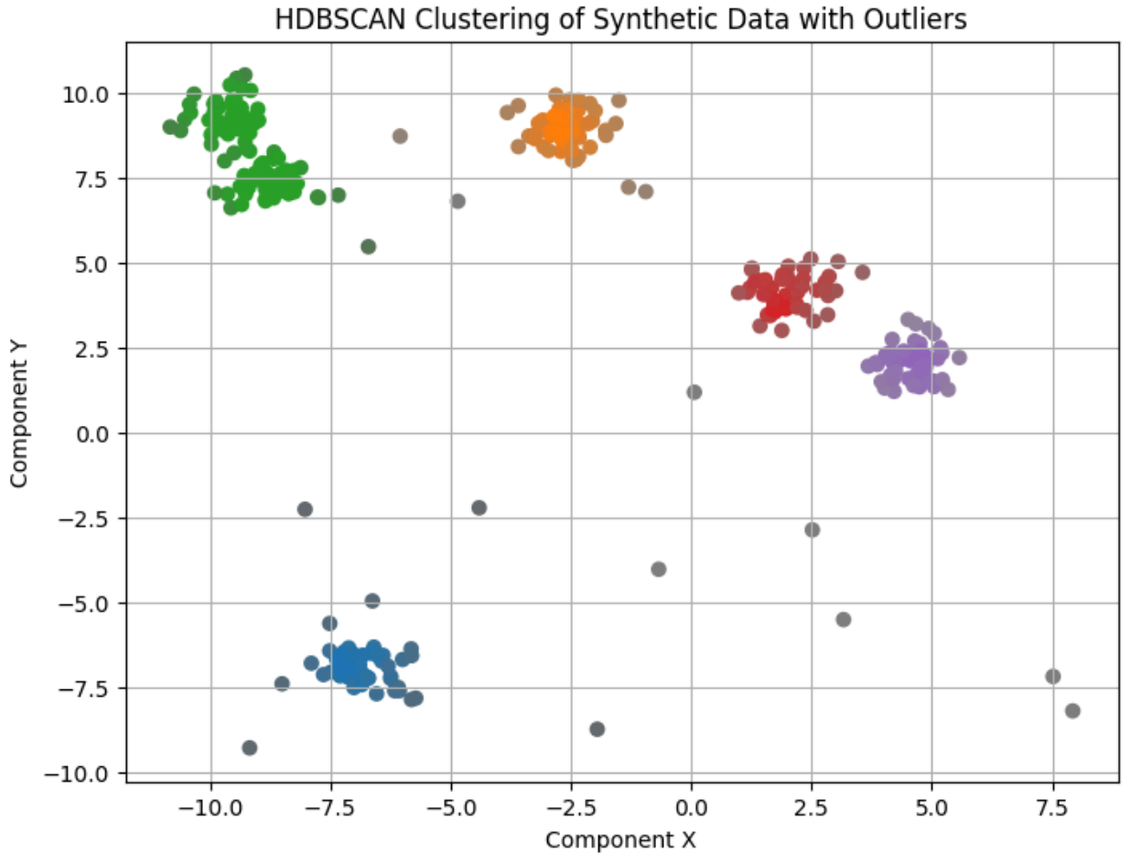


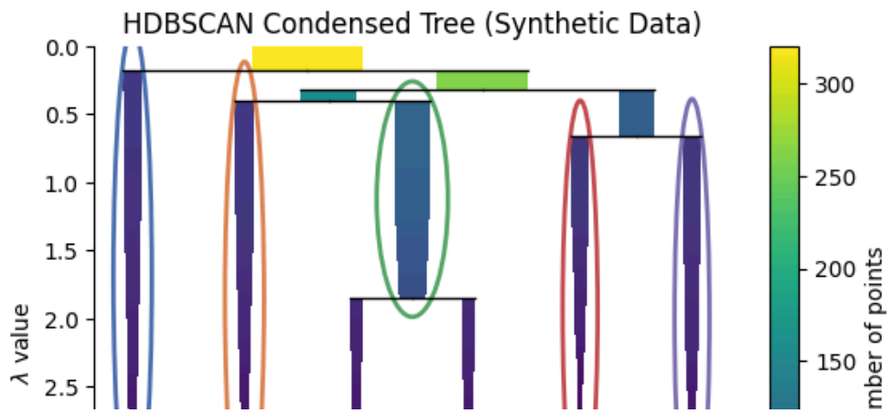


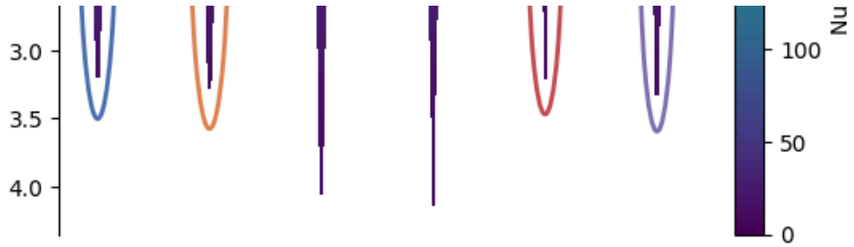

```
import hdbscan
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.datasets import make_blobs

import warnings
warnings.filterwarnings("ignore")

# Generate synthetic 2D data with 6 clusters
X, _ = make_blobs(n_samples=300, centers=6, n_features=2, cluster_std=0.5,
                  random_state=42)

# Add some random noise/outliers
outliers = np.random.uniform(low=-10, high=10, size=(20, 2))
X = np.vstack([X, outliers])

# Save to CSV
df = pd.DataFrame(X, columns=["X", "Y"])
df.to_csv("synthetic_clusters.csv", index=False, header=False)

# Load the projection
projection = np.loadtxt("synthetic_clusters.csv", delimiter=",")

# Convert to list (optional step)
lis = projection.tolist()
print(f"Data preview (first 10 points):\n{lis[:10]}")

#-----------------------------------Clustering-------------------------------------------------
clusterer = hdbscan.HDBSCAN(min_cluster_size=8, gen_min_span_tree=True)
clusterer.fit(projection)

# Color mapping
palette = sns.color_palette()
cluster_colors = [sns.desaturate(palette[col % len(palette)], sat)
                  if col >= 0 else (0.5, 0.5, 0.5) for col, sat in
```

```
                   zip(clusterer.labels_, clusterer.probabilities_)]

# Scatter plot
plt.figure(figsize=(8, 6))
plt.scatter(projection.T[0], projection.T[1], c=cluster_colors)
plt.title('HDBSCAN Clustering of Synthetic Data with Outliers')
plt.xlabel('Component X')
plt.ylabel('Component Y')
plt.grid(True)
plt.show()

# Condensed tree plot
clusterer.condensed_tree_.plot(select_clusters=True, selection_palette=sns.color_palette('deep', 12))
plt.title('HDBSCAN Condensed Tree (Synthetic Data)')
plt.show()
```

```
Data preview (first 10 points):
[[-1.566104672447485, 9.10157503461424], [-10.327603236973872, 9.97007406484335], [-2.4047658255503723, 8.034451066258434], [4.528147443565176, 2.330169930986778],
```





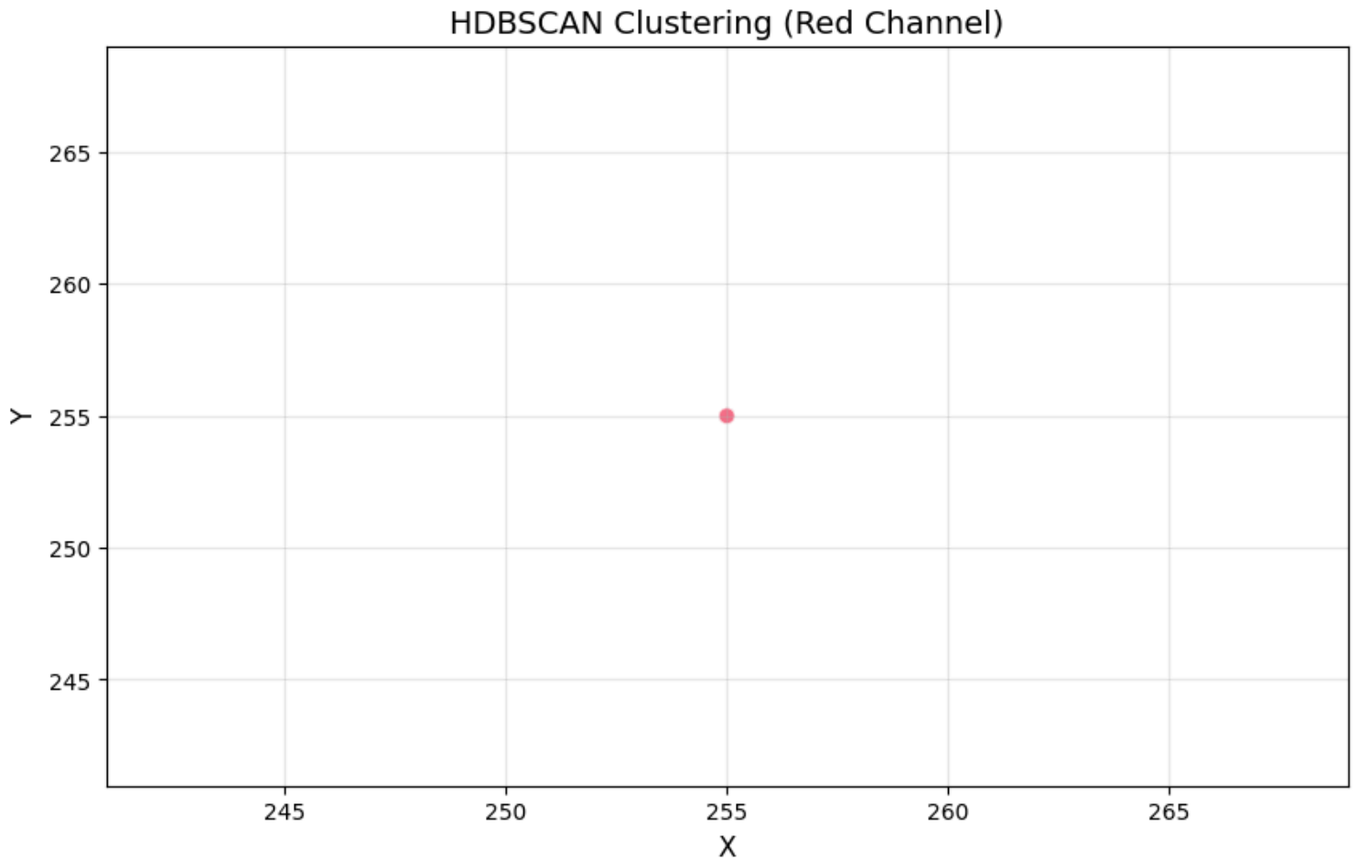


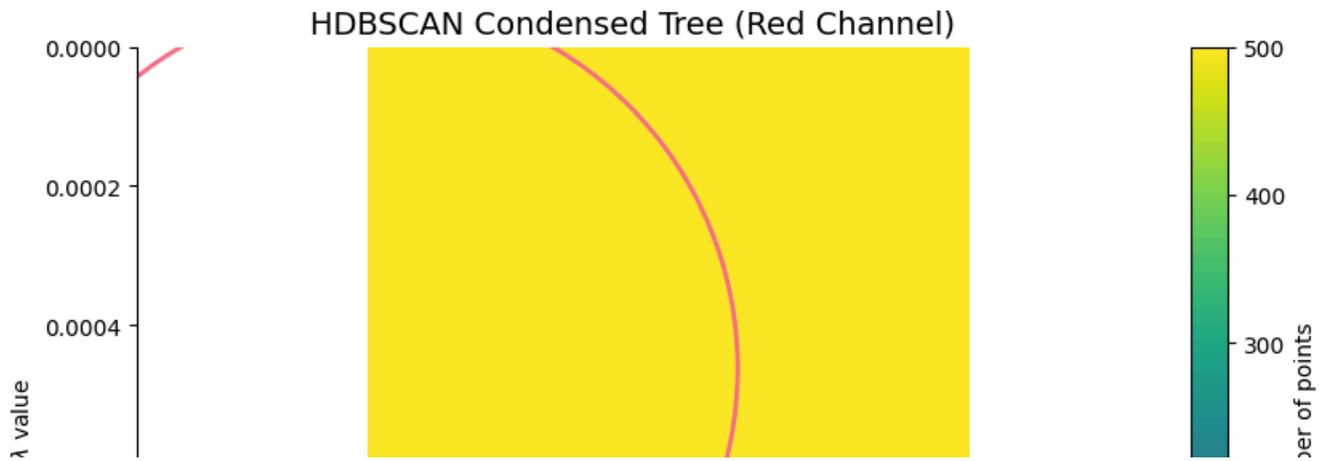
Deconvoluted Images

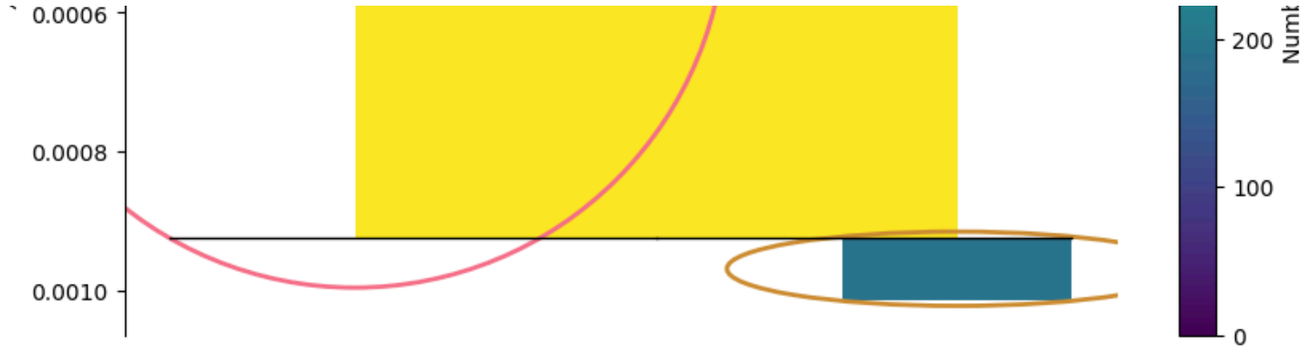
White Square Black Circle

```
import hdbscan
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

import warnings
warnings.filterwarnings("ignore")

# --- Load Data ---
#projection = pd.read_csv("WhiteSquare_Black_Circle_R.csv")
projection = np.loadtxt("WhiteSquare_Black_Circle_R.csv", delimiter=",")

# --- Clustering ---
clusterer = hdbscan.HDBSCAN(
    min_cluster_size=190,
    min_samples=5,
    cluster_selection_method='eom',
    gen_min_span_tree=True
)
clusterer.fit(projection)

# --- Plot Clusters ---
palette = sns.color_palette("husl", np.unique(clusterer.labels_).max() + 1)
cluster_colors = [
```

```
    palette[col] if col >= 0 else (0.9, 0.9, 0.9)
    for col in clusterer.labels_
]

plt.figure(figsize=(10, 6))
plt.scatter(
    projection[:, 0], projection[:, 1],
    c=cluster_colors,
    s=50,
    alpha=0.7,
    edgecolor='w',
    linewidth=0.5
)
plt.title('HDBSCAN Clustering (Red Channel)', fontsize=14)
plt.xlabel('X', fontsize=12)
plt.ylabel('Y', fontsize=12)
plt.grid(True, alpha=0.3)
plt.show()

# --- Condensed Tree (for Stability Analysis) ---
plt.figure(figsize=(10, 6))
clusterer.condensed_tree_.plot(
    select_clusters=True,
    selection_palette=sns.color_palette("husl", 8),
)
plt.title('HDBSCAN Condensed Tree (Red Channel)', fontsize=14)
plt.show()

# --- Additional Diagnostics ---
# Print cluster statistics
cluster_stats = pd.Series(clusterer.labels_).value_counts()
print("\nCluster Membership:")
print(cluster_stats)

# Check % of noise points
noise_ratio = (clusterer.labels_ == -1).mean() * 100
print(f"\nNoise Points: {noise_ratio:.2f}%")
```


HDBSCAN Clustering (Red Channel)
265
260
255
250
245
Y
245
250
255
260
265
X

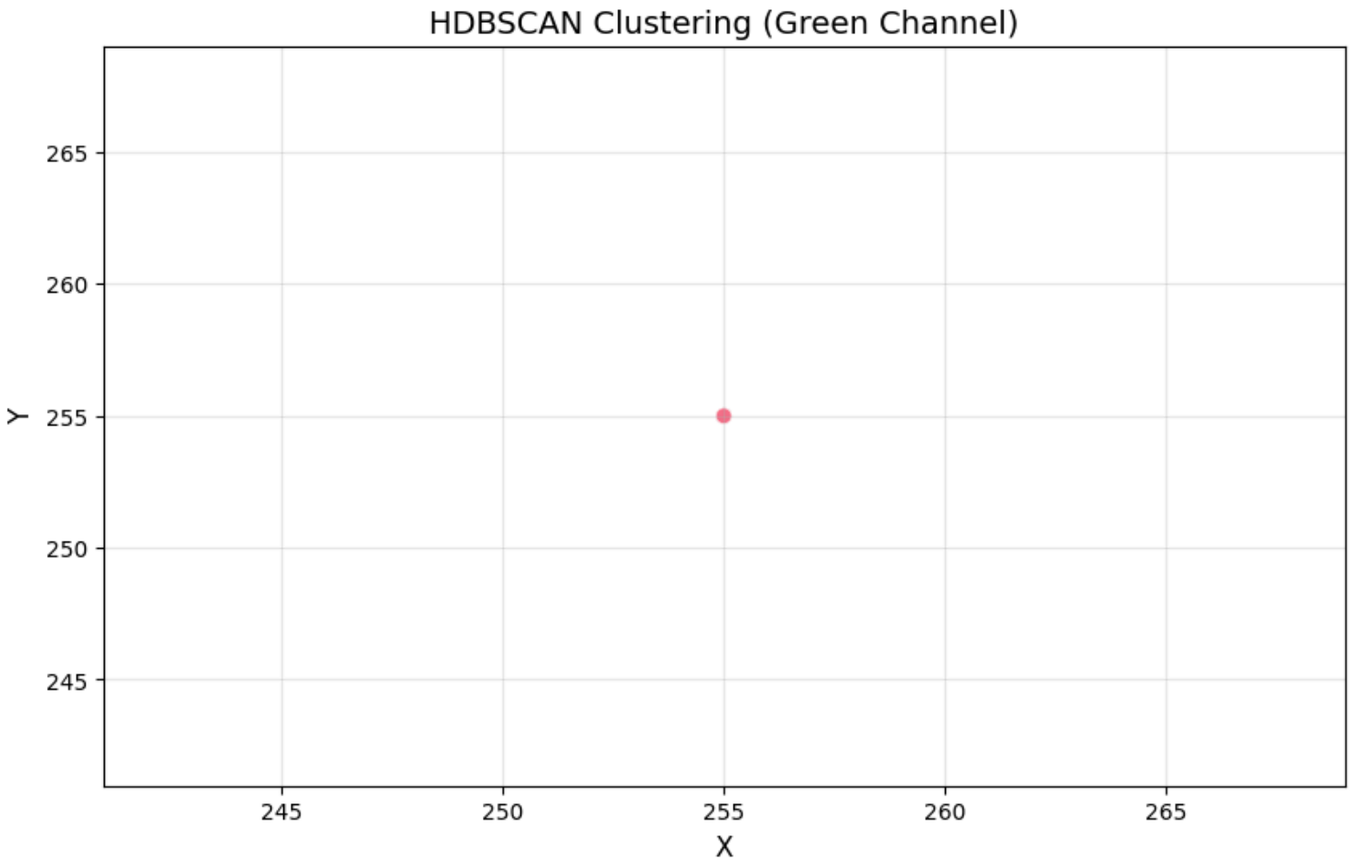

HDBSCAN Condensed Tree (Red Channel)
0.0000
0.0002
0.0004
500
400
300

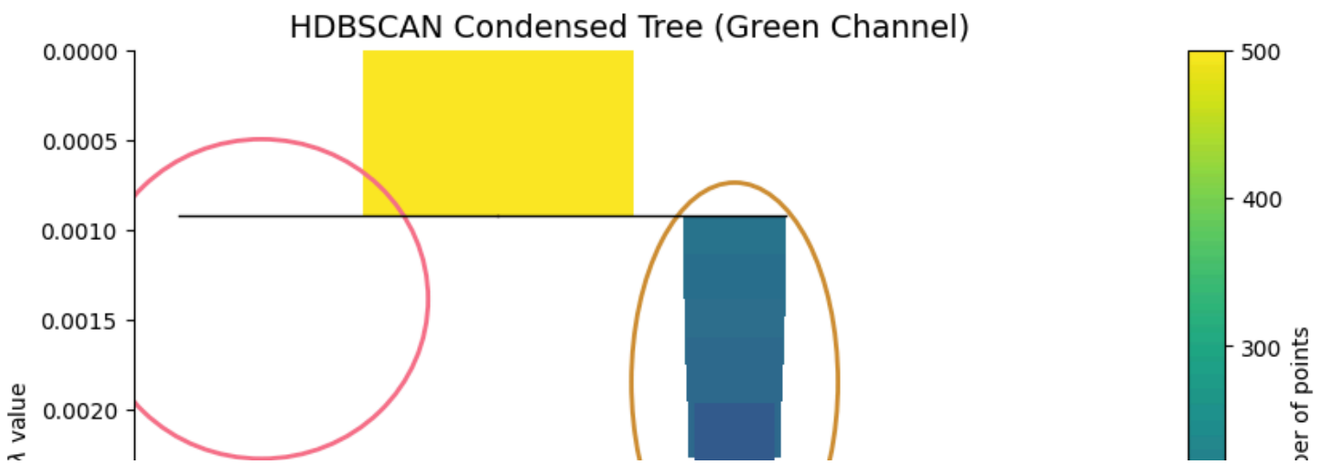


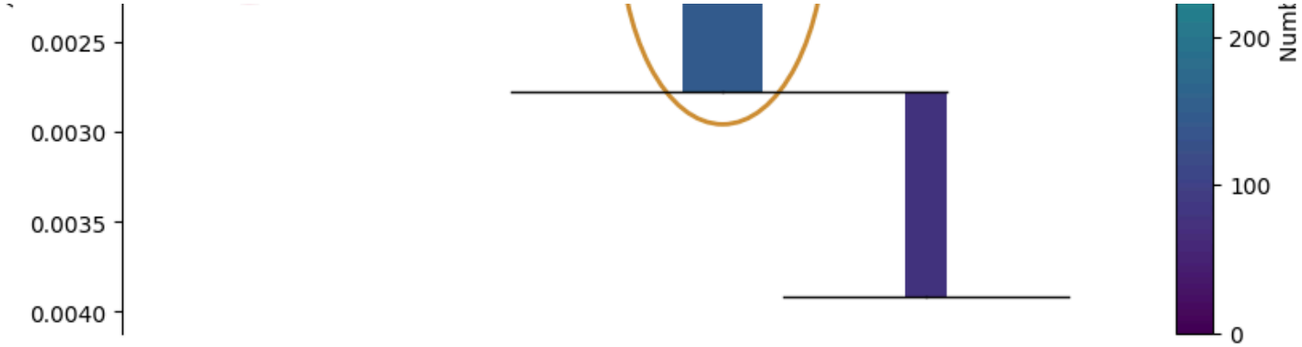

```
Cluster Membership:
 0    308
 1    191
-1      1
Name: count, dtype: int64

Noise Points: 0.20%
```

```
import hdbscan
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

import warnings
warnings.filterwarnings("ignore")

# --- Load Data ---
projection = np.loadtxt("WhiteSquare_Black_Circle_G.csv", delimiter=",")

# --- Clustering ---
clusterer = hdbscan.HDBSCAN(
    min_cluster_size=20,
    min_samples=5,
    cluster_selection_method='eom',
    gen_min_span_tree=True
)
clusterer.fit(projection)

# --- Plot Clusters ---
palette = sns.color_palette("husl", np.unique(clusterer.labels_).max() + 1)
```

```
cluster_colors = [
    palette[col] if col >= 0 else (0.9, 0.9, 0.9)
    for col in clusterer.labels_
]

plt.figure(figsize=(10, 6))
plt.scatter(
    projection[:, 0], projection[:, 1],
    c=cluster_colors,
    s=50,
    alpha=0.7,
    edgecolor='w',
    linewidth=0.5
)
plt.title('HDBSCAN Clustering (Green Channel)', fontsize=14)
plt.xlabel('X', fontsize=12)
plt.ylabel('Y', fontsize=12)
plt.grid(True, alpha=0.3)
plt.show()

# --- Condensed Tree (Cluster Stability) ---
plt.figure(figsize=(10, 6))
clusterer.condensed_tree_.plot(
    select_clusters=True,
    selection_palette=sns.color_palette("husl", 8),
)
plt.title('HDBSCAN Condensed Tree (Green Channel)', fontsize=14)
plt.show()

# --- Diagnostics ---
# Cluster membership counts
cluster_stats = pd.Series(clusterer.labels_).value_counts()
print("\nCluster Membership:")
print(cluster_stats)

# Noise ratio
noise_ratio = (clusterer.labels_ == -1).mean() * 100
print(f"\nNoise Points: {noise_ratio:.2f}%")
```


HDBSCAN Clustering (Green Channel)
265
260
255
250
245
Y
245
250
255
260
265
X

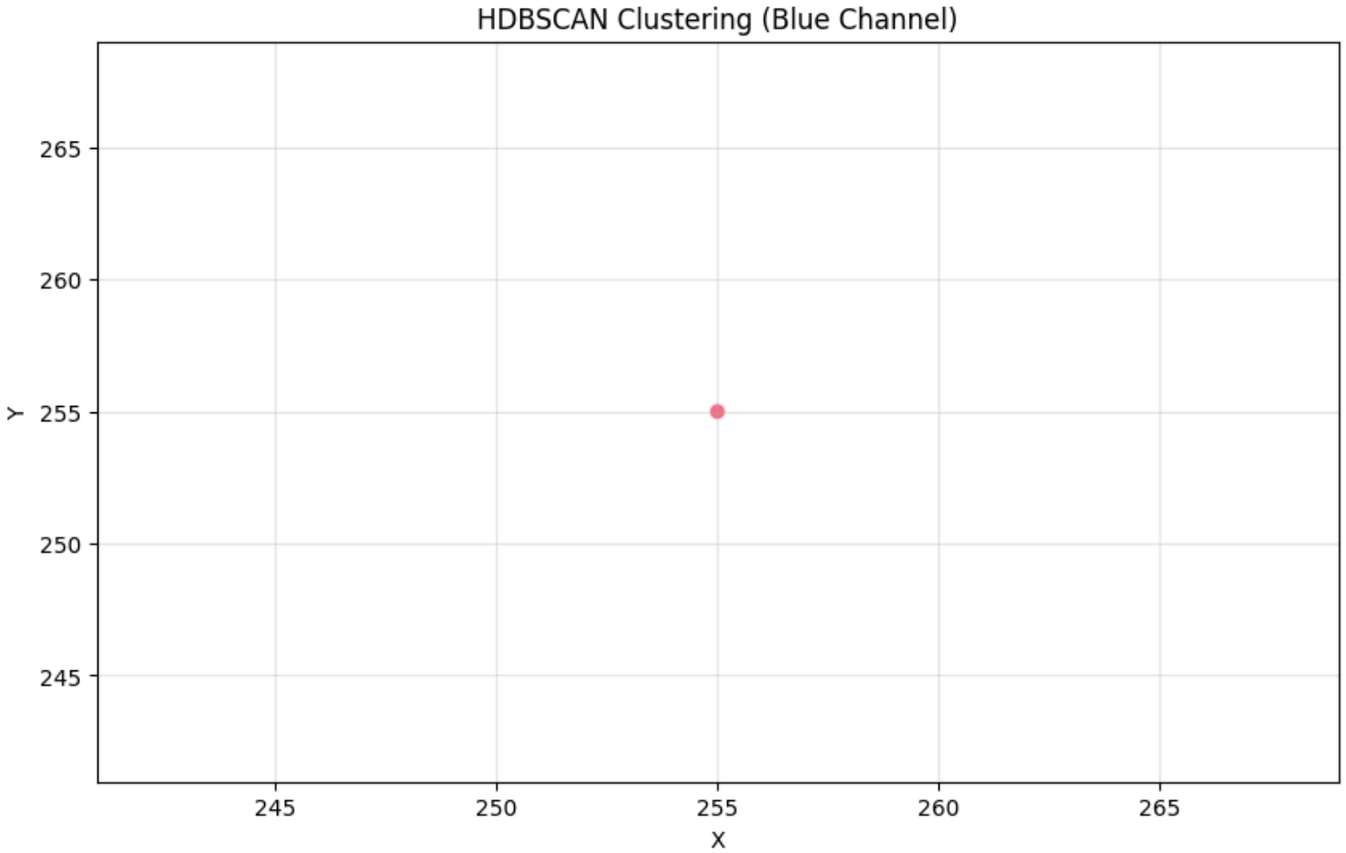

HDBSCAN Condensed Tree (Green Channel)
0.0000
0.0005
0.0010
0.0015
0.0020
λ value
500
400
300
er of points

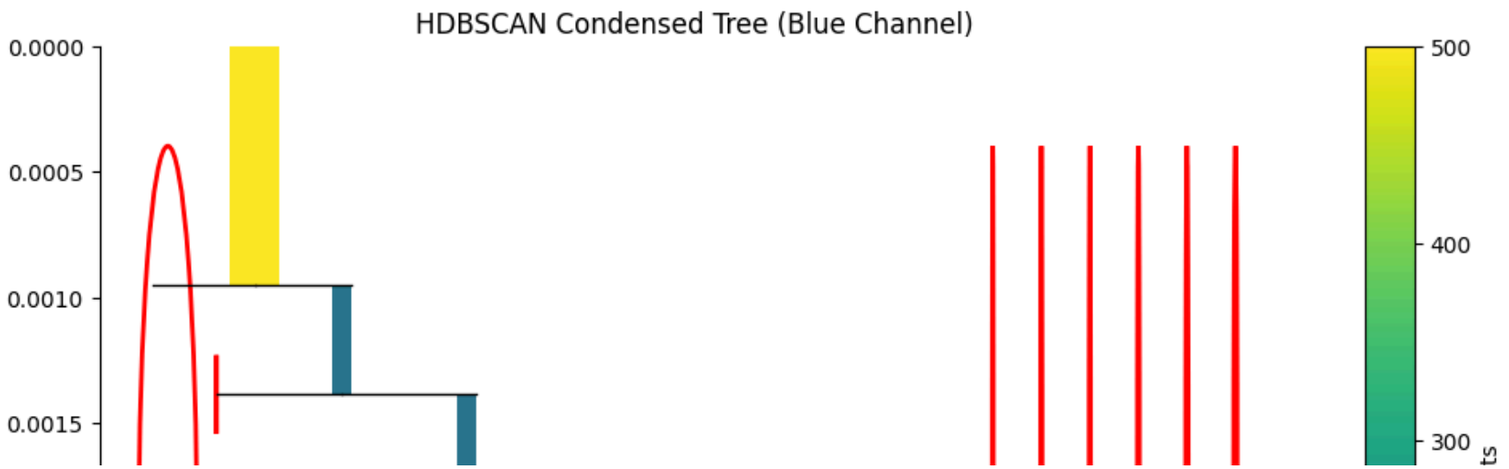


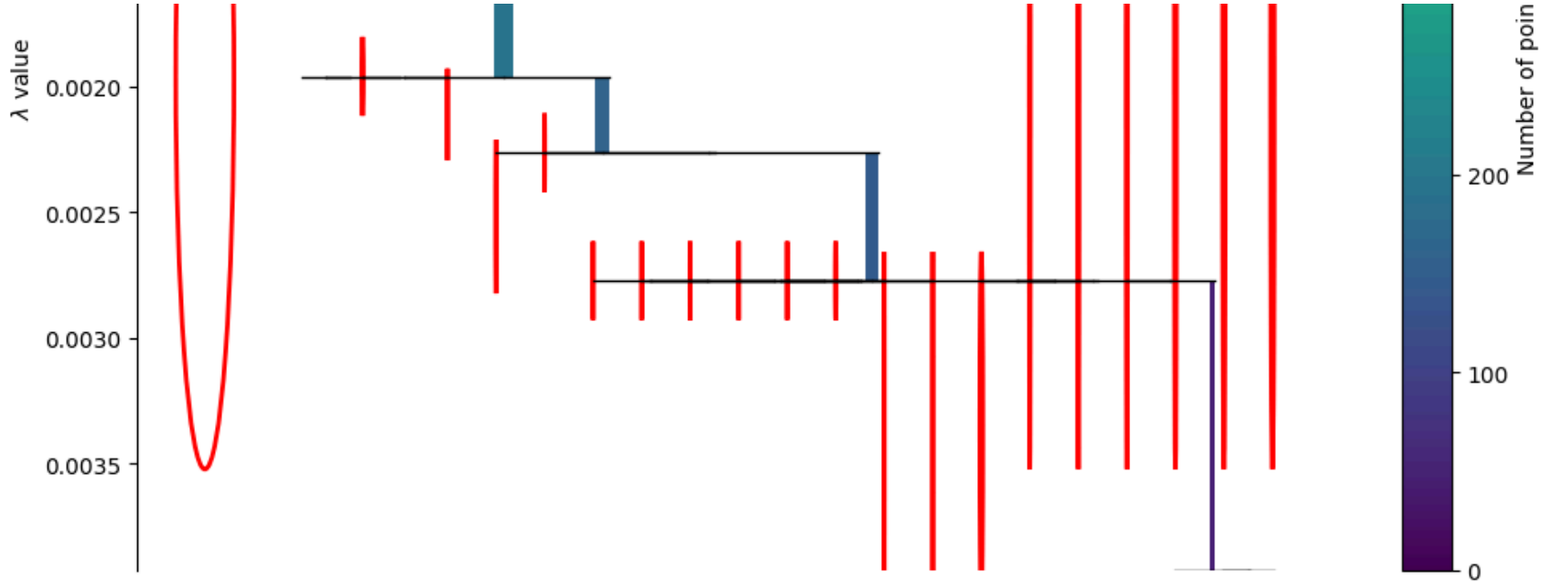

```
Cluster Membership:
 0    308
 1    191
-1      1
Name: count, dtype: int64

Noise Points: 0.20%
```

```
import hdbscan
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

import warnings
warnings.filterwarnings("ignore")

# Load the data
projection = np.loadtxt("WhiteSquare_Black_Circle_B.csv", delimiter=",")
projection_t = projection.T

# Run HDBSCAN clustering
clusterer = hdbscan.HDBSCAN(
    min_cluster_size=2,
    min_samples=5,
    cluster_selection_method='eom',
    gen_min_span_tree=True
)
clusterer.fit(projection_t)

# Plot clusters
```

```
palette = sns.color_palette("husl", np.unique(clusterer.labels_).max() + 1)
cluster_colors = [palette[col] if col >= 0 else (0.9, 0.9, 0.9) for col in clusterer.labels_]

plt.figure(figsize=(10, 6))
plt.scatter(
    projection[:, 0], projection[:, 1],
    c=cluster_colors,
    s=50,
    alpha=0.7,
    edgecolor='w',
    linewidth=0.5
)
plt.title('HDBSCAN Clustering (Blue Channel)')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid(True, alpha=0.3)
plt.show()

# Plot condensed tree
plt.figure(figsize=(12, 8))
clusterer.condensed_tree_.plot(
    select_clusters=True,
    selection_palette=sns.color_palette("husl", 8)
)
plt.title('HDBSCAN Condensed Tree (Blue Channel)')
plt.show()

# Basic cluster stats
print("\nCluster Membership:")
print(pd.Series(clusterer.labels_).value_counts())
print(f"\nNoise Points: {(clusterer.labels_ == -1).mean()*100:.2f}%")
```


HDBSCAN Clustering (Blue Channel)
265
260
255
250
245
Y
245
250
255
260
265
X

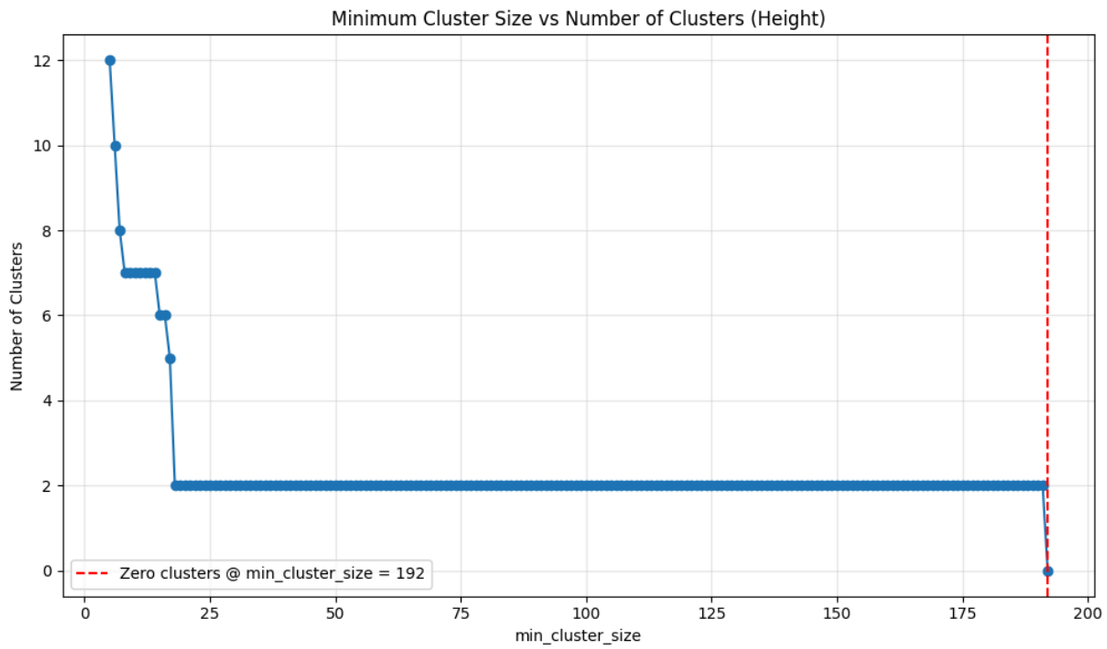

HDBSCAN Condensed Tree (Blue Channel)
0.0000
0.0005
0.0010
0.0015
500
400
300
ts

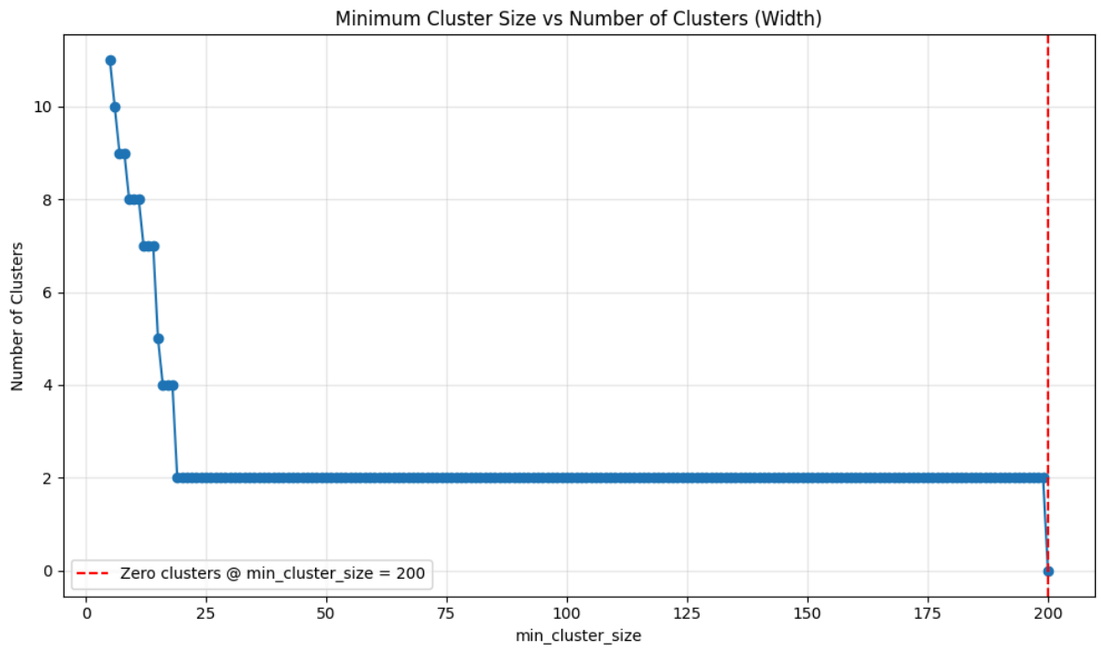



```
Cluster Membership:
 0     300
-1      74
 12     19
 6      16
 11     14
 10     11
 3       9
 7       8
 9       8
 8       6
 4       5
 2       5
 19      5
 5       3
 17      3
 1       2
 18      2
 15      2
 13      2
 16      2
 14      2
 20      2
Name: count, dtype: int64

Noise Points: 14.80%
```

```
import hdbscan
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

# Load the data
projection = np.loadtxt("/content/WhiteSquare_Black_Circle_R - WhiteSquare_Black_Circle_R.csv",
                        delimiter=",")
projection_t = projection.T

# Create figure with two subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6))

# First plot (Height)
# Initialize tracking lists
min_cluster_sizes_height = []
n_clusters_height = []
# Start sweep
size = 5
while True:
    clusterer = hdbscan.HDBSCAN(
        min_cluster_size=size,
        min_samples=5,
        cluster_selection_method='eom'
    )
    clusterer.fit(projection)
    labels = clusterer.labels_
    num_clusters = len(set(labels) - {-1})

    min_cluster_sizes_height.append(size)
    n_clusters_height.append(num_clusters)

    if num_clusters == 0:
        break
    size += 1

ax1.plot(min_cluster_sizes_height, n_clusters_height, marker='o', linestyle='-')
ax1.axvline(size, color='red', linestyle='--', label=f"Zero clusters @ min_cluster_size = {size}")
ax1.set_title('Minimum Cluster Size vs Number of Clusters (Height)')
ax1.set_xlabel('min_cluster_size')
ax1.set_ylabel('Number of Clusters')
ax1.legend()
ax1.grid(True, alpha=0.3)
```

```
# Second plot (Width)
# Initialize tracking lists
min_cluster_sizes_width = []
n_clusters_width = []
# Start sweep
size = 5
while True:
    clusterer = hdbscan.HDBSCAN(
        min_cluster_size=size,
        min_samples=5,
        cluster_selection_method='eom'
    )
    clusterer.fit(projection_t)
    labels = clusterer.labels_
    num_clusters = len(set(labels) - {-1})

    min_cluster_sizes_width.append(size)
    n_clusters_width.append(num_clusters)

    if num_clusters == 0:
        break
    size += 1

ax2.plot(min_cluster_sizes_width, n_clusters_width, marker='o', linestyle='-')
ax2.axvline(size, color='red', linestyle='--', label=f"Zero clusters @ min_cluster_size = {size}")
ax2.set_title('Minimum Cluster Size vs Number of Clusters (Width)')
ax2.set_xlabel('min_cluster_size')
ax2.set_ylabel('Number of Clusters')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

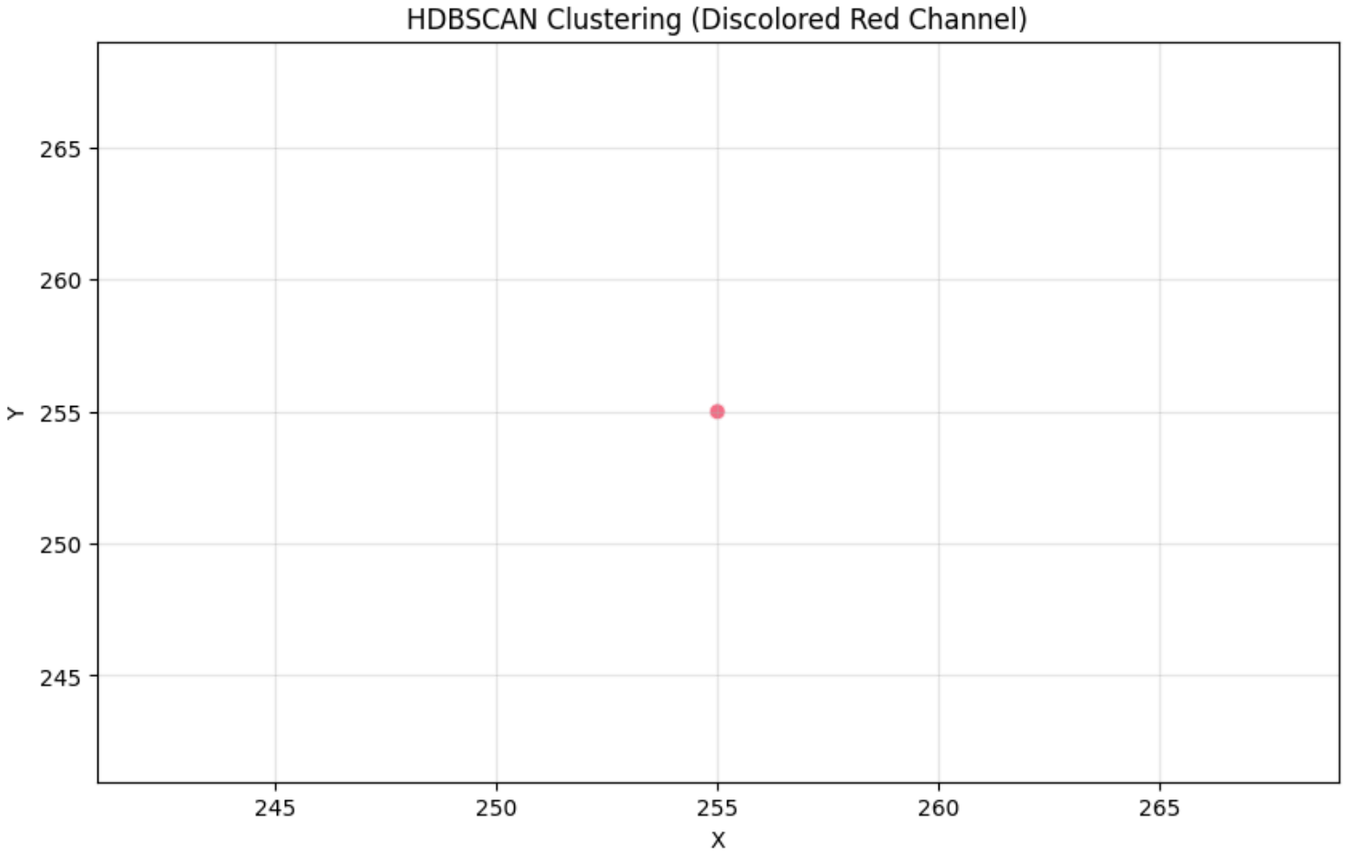




White Square Black Circle Discolored

```
import hdbscan
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

import warnings
warnings.filterwarnings("ignore")

# Load the data
projection = np.loadtxt("/content/whitesquareblackcirclediscoloredR.csv", delimiter=",")

# Run HDBSCAN clustering
clusterer = hdbscan.HDBSCAN(
    min_cluster_size=10,
    min_samples=5,
```

```
    cluster_selection_method='eom',
    gen_min_span_tree=True
)
clusterer.fit(projection)

# Plot clusters
palette = sns.color_palette("husl", np.unique(clusterer.labels_).max() + 1)
cluster_colors = [palette[col] if col >= 0 else (0.9, 0.9, 0.9) for col in clusterer.labels_]

plt.figure(figsize=(10, 6))
plt.scatter(
    projection[:, 0], projection[:, 1],
    c=cluster_colors,
    s=50,
    alpha=0.7,
    edgecolor='w',
    linewidth=0.5
)
plt.title('HDBSCAN Clustering (Discolored Red Channel)')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid(True, alpha=0.3)
plt.show()

# Plot condensed tree
plt.figure(figsize=(10, 6))
clusterer.condensed_tree_.plot(
    select_clusters=True,
    selection_palette=sns.color_palette("husl", 8)
)
plt.title('HDBSCAN Condensed Tree (Discolored Red Channel)')
plt.show()

# Basic cluster stats
print("\nCluster Membership:")
print(pd.Series(clusterer.labels_).value_counts())
print(f"\nNoise Points: {(clusterer.labels_ == -1).mean()*100:.2f}%")
```

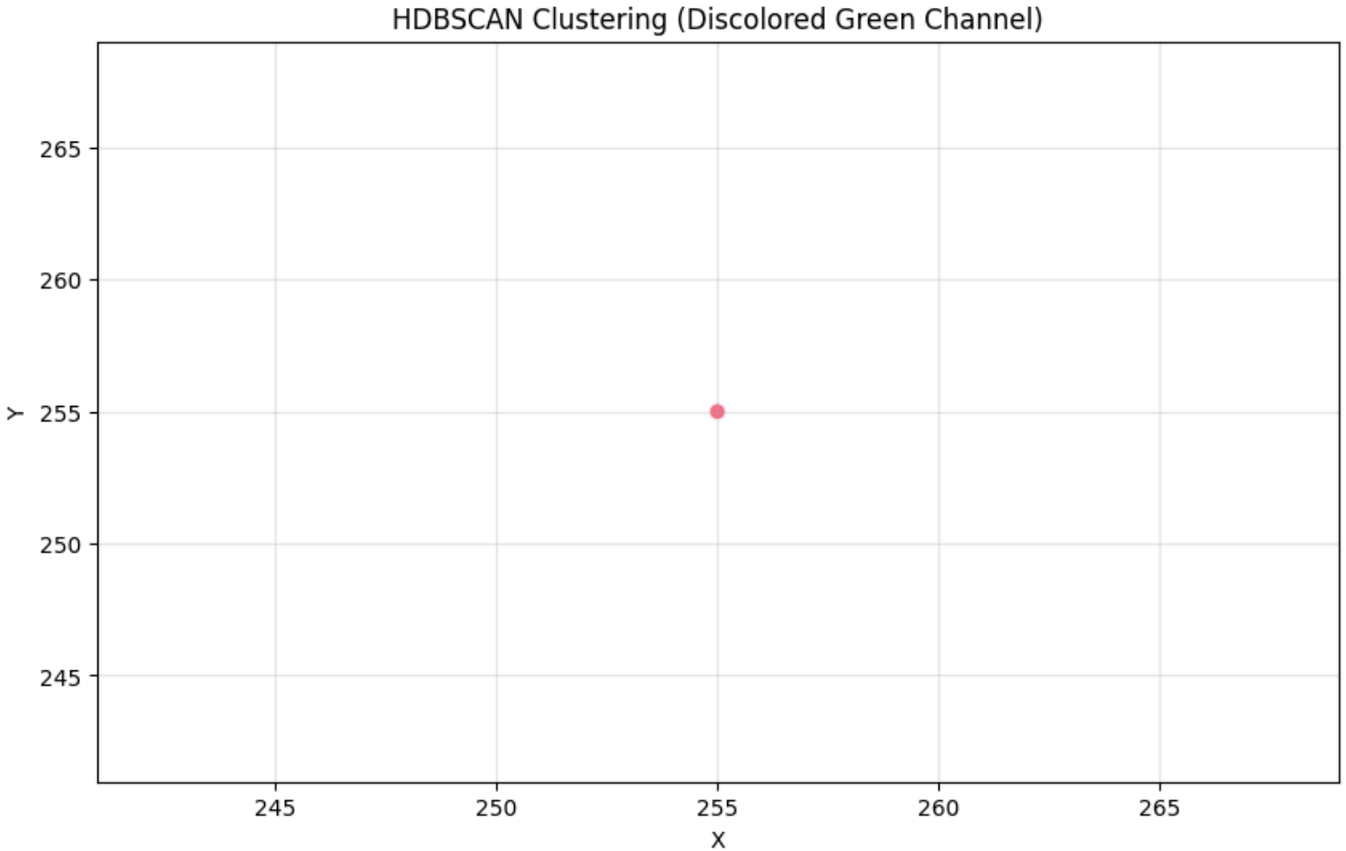

HDBSCAN Clustering (Discolored Red Channel)
265
260
255
250
245
Y
245
250
255
260
265
X

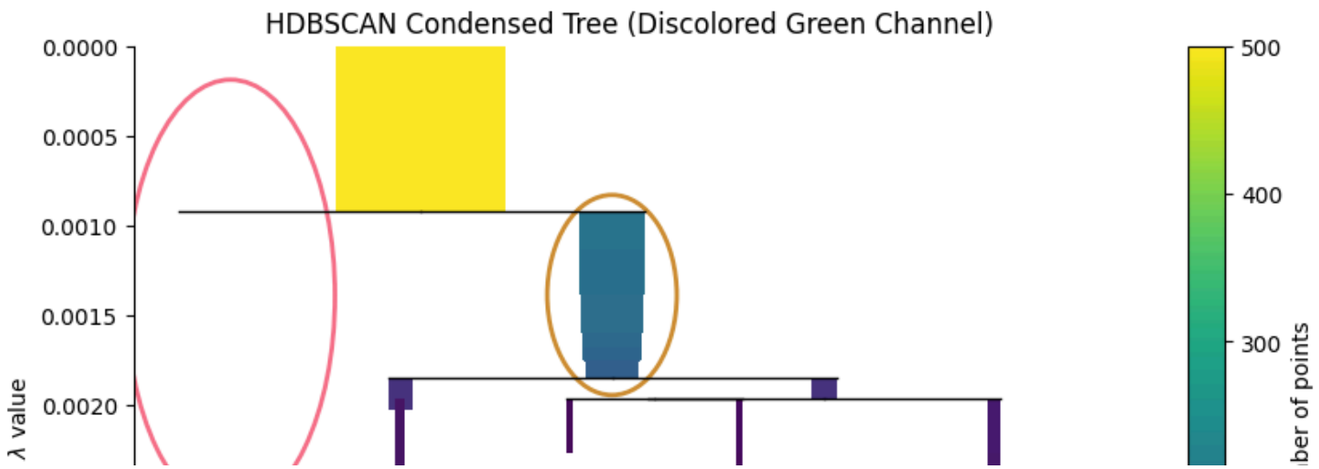

HDBSCAN Condensed Tree (Discolored Red Channel)
0.0000
0.0005
0.0010
0.0015
0.0020
λ value
500
400
300
ber of points


```
Cluster Membership:
 0    308
 1    191
-1      1
Name: count, dtype: int64

Noise Points: 0.20%
```

```
import hdbscan
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

import warnings
warnings.filterwarnings("ignore")

# Load the data
projection = np.loadtxt("/content/whitesquareblackcirclediscoloredG.csv", delimiter=",")

# Run HDBSCAN clustering
clusterer = hdbscan.HDBSCAN(
    min_cluster_size=10,
    min_samples=5,
    cluster_selection_method='eom',
    gen_min_span_tree=True
)
clusterer.fit(projection)

# Plot clusters
palette = sns.color_palette("husl", np.unique(clusterer.labels_).max() + 1)
```

```
cluster_colors = [palette[col] if col >= 0 else (0.9, 0.9, 0.9) for col in clusterer.labels_]

plt.figure(figsize=(10, 6))
plt.scatter(
    projection[:, 0], projection[:, 1],
    c=cluster_colors,
    s=50,
    alpha=0.7,
    edgecolor='w',
    linewidth=0.5
)
plt.title('HDBSCAN Clustering (Discolored Green Channel)')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid(True, alpha=0.3)
plt.show()

# Plot condensed tree
plt.figure(figsize=(10, 6))
clusterer.condensed_tree_.plot(
    select_clusters=True,
    selection_palette=sns.color_palette("husl", 8)
)
plt.title('HDBSCAN Condensed Tree (Discolored Green Channel)')
plt.show()

# Basic cluster stats
print("\nCluster Membership:")
print(pd.Series(clusterer.labels_).value_counts())
print(f"\nNoise Points: {(clusterer.labels_ == -1).mean()*100:.2f}%")
```

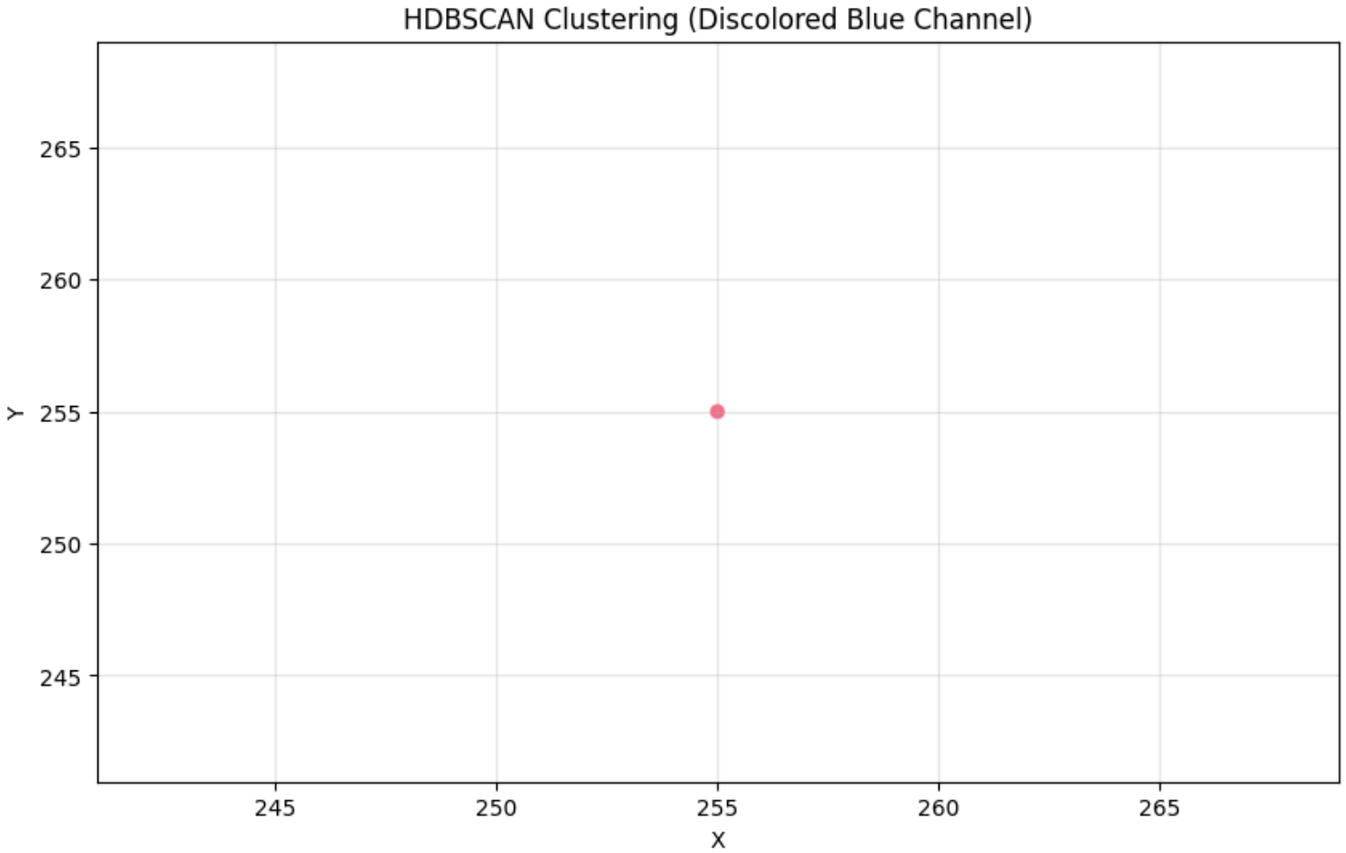

HDBSCAN Clustering (Discolored Green Channel)
Y
X
265
260
255
250
245
245
250
255
260
265

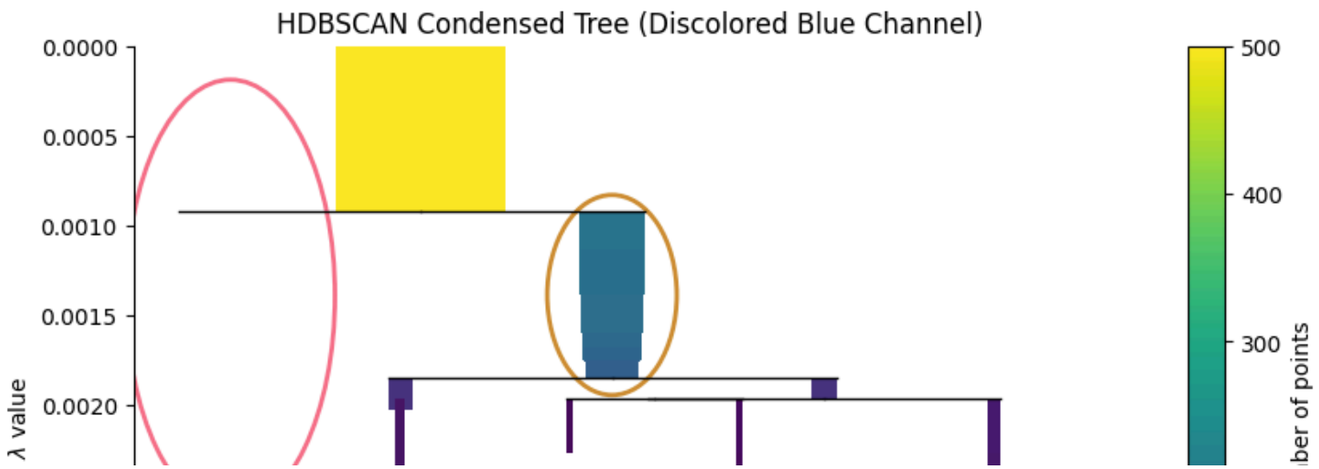

HDBSCAN Condensed Tree (Discolored Green Channel)
λ value
0.0000
0.0005
0.0010
0.0015
0.0020
500
400
300
ber of points


```
Cluster Membership:
 0    308
 1    191
-1      1
Name: count, dtype: int64

Noise Points: 0.20%
```

```
import hdbscan
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

import warnings
warnings.filterwarnings("ignore")

# Load the data
projection = np.loadtxt("/content/whitesquareblackcirclediscoloredB.csv", delimiter=",")

# Run HDBSCAN clustering
clusterer = hdbscan.HDBSCAN(
    min_cluster_size=10,
    min_samples=5,
    cluster_selection_method='eom',
    gen_min_span_tree=True
)
clusterer.fit(projection)

# Plot clusters
palette = sns.color_palette("husl", np.unique(clusterer.labels_).max() + 1)
```

```
cluster_colors = [palette[col] if col >= 0 else (0.9, 0.9, 0.9) for col in clusterer.labels_]

plt.figure(figsize=(10, 6))
plt.scatter(
    projection[:, 0], projection[:, 1],
    c=cluster_colors,
    s=50,
    alpha=0.7,
    edgecolor='w',
    linewidth=0.5
)
plt.title('HDBSCAN Clustering (Discolored Blue Channel)')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid(True, alpha=0.3)
plt.show()

# Plot condensed tree
plt.figure(figsize=(10, 6))
clusterer.condensed_tree_.plot(
    select_clusters=True,
    selection_palette=sns.color_palette("husl", 8)
)
plt.title('HDBSCAN Condensed Tree (Discolored Blue Channel)')
plt.show()

# Basic cluster stats
print("\nCluster Membership:")
print(pd.Series(clusterer.labels_).value_counts())
print(f"\nNoise Points: {(clusterer.labels_ == -1).mean()*100:.2f}%")
```


HDBSCAN Clustering (Discolored Blue Channel)
265
260
255
250
245
Y
245
250
255
260
265
X

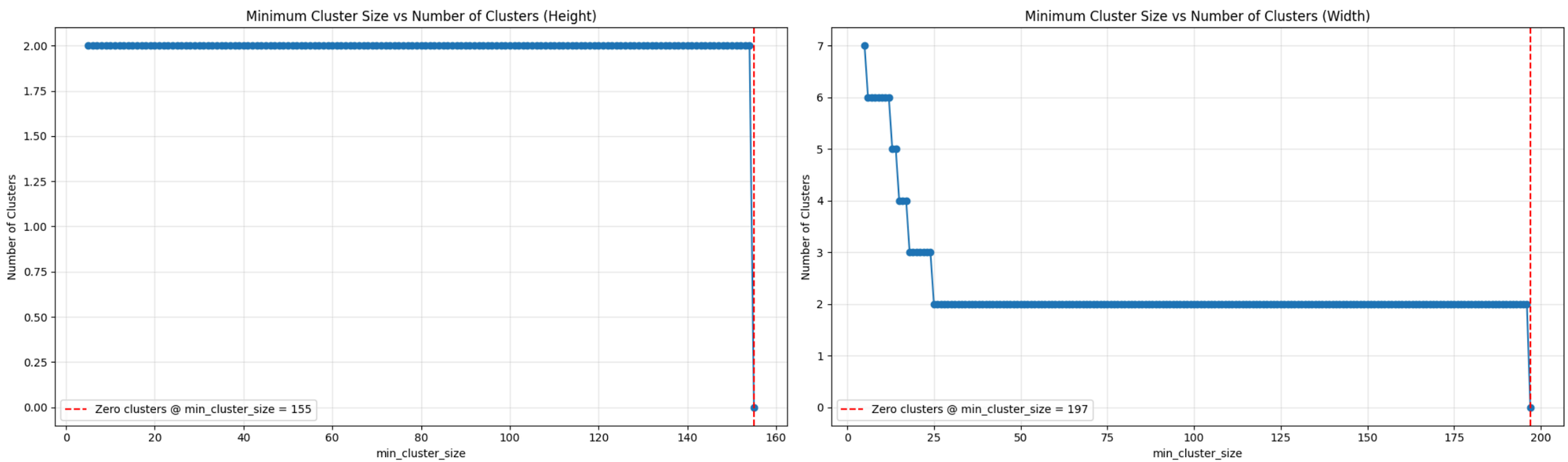

HDBSCAN Condensed Tree (Discolored Blue Channel)
0.0000
0.0005
0.0010
0.0015
0.0020
λ value
500
400
300
ber of points


```
Cluster Membership:
 0    308
 1    191
-1      1
Name: count, dtype: int64

Noise Points: 0.20%
```

```
import hdbscan
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

# Load the data
projection = np.loadtxt("/content/BlackRectangle_B.csv",
                        delimiter=",")
projection_t = projection.T

# Create figure with two subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6))

# First plot (Height)
# Initialize tracking lists
min_cluster_sizes_height = []
n_clusters_height = []
# Start sweep
size = 5
while True:
    clusterer = hdbscan.HDBSCAN(
        min_cluster_size=size,
```

```
        min_samples=5,
        cluster_selection_method='eom'
    )
    clusterer.fit(projection)
    labels = clusterer.labels_
    num_clusters = len(set(labels) - {-1})

    min_cluster_sizes_height.append(size)
    n_clusters_height.append(num_clusters)

    if num_clusters == 0:
        break
    size += 1

ax1.plot(min_cluster_sizes_height, n_clusters_height, marker='o', linestyle='-')
ax1.axvline(size, color='red', linestyle='--', label=f"Zero clusters @ min_cluster_size = {size}")
ax1.set_title('Minimum Cluster Size vs Number of Clusters (Height)')
ax1.set_xlabel('min_cluster_size')
ax1.set_ylabel('Number of Clusters')
ax1.legend()
ax1.grid(True, alpha=0.3)

# Second plot (Width)
# Initialize tracking lists
min_cluster_sizes_width = []
n_clusters_width = []
# Start sweep
size = 5
while True:
    clusterer = hdbscan.HDBSCAN(
        min_cluster_size=size,
        min_samples=5,
        cluster_selection_method='eom'
    )
    clusterer.fit(projection_t)
    labels = clusterer.labels_
    num_clusters = len(set(labels) - {-1})

    min_cluster_sizes_width.append(size)
    n_clusters_width.append(num_clusters)

    if num_clusters == 0:
        break
    size += 1
```

```
ax2.plot(min_cluster_sizes_width, n_clusters_width, marker='o', linestyle='-')
ax2.axvline(size, color='red', linestyle='--', label=f"Zero clusters @ min_cluster_size = {size}")
ax2.set_title('Minimum Cluster Size vs Number of Clusters (Width)')
ax2.set_xlabel('min_cluster_size')
ax2.set_ylabel('Number of Clusters')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

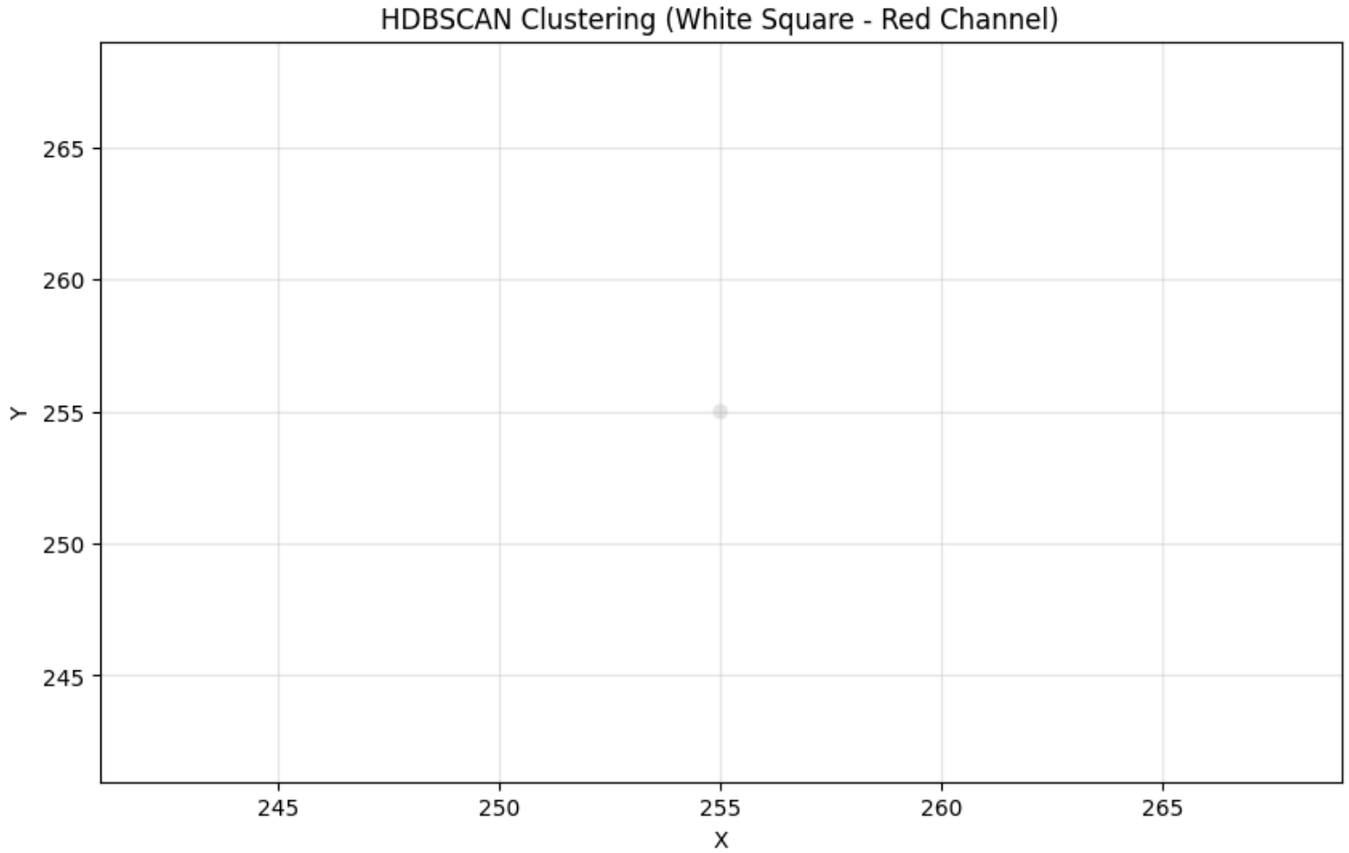


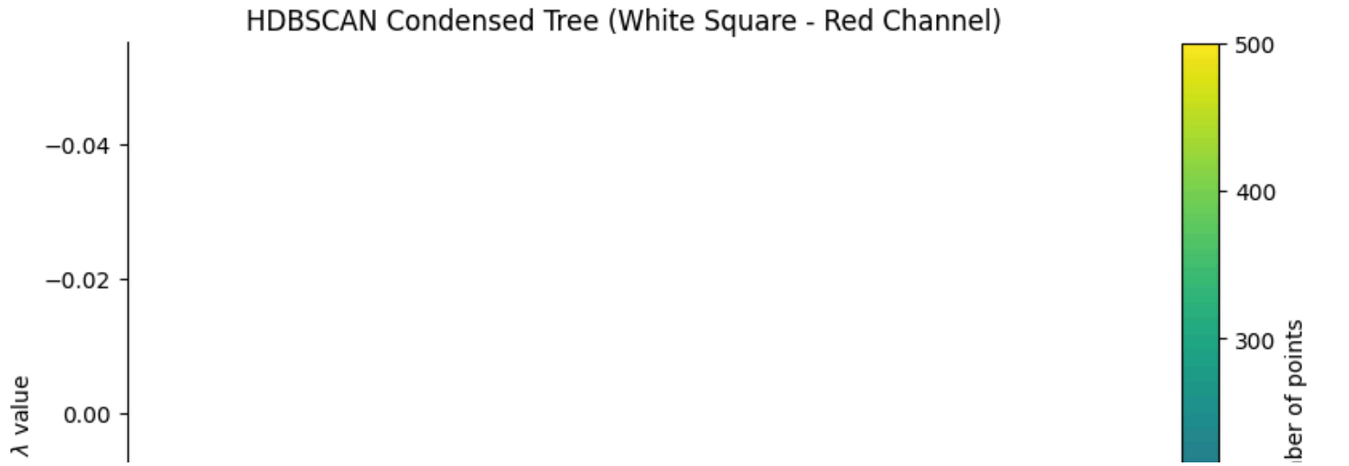
White Square

```
import hdbscan
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
import pandas as pd

import warnings
warnings.filterwarnings("ignore")

# Load the data
projection = np.loadtxt("/content/whitesquareR.csv", delimiter=",")

# Run HDBSCAN clustering
clusterer = hdbscan.HDBSCAN(
    min_cluster_size=20,
    min_samples=5,
    cluster_selection_method='eom',
    gen_min_span_tree=True
)
clusterer.fit(projection)

# Plot clusters
palette = sns.color_palette("husl", np.unique(clusterer.labels_).max() + 1)
cluster_colors = [palette[col] if col >= 0 else (0.9, 0.9, 0.9) for col in clusterer.labels_]

plt.figure(figsize=(10, 6))
plt.scatter(
    projection[:, 0], projection[:, 1],
    c=cluster_colors,
    s=50,
    alpha=0.7,
    edgecolor='w',
    linewidth=0.5
)
plt.title('HDBSCAN Clustering (White Square - Red Channel)')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid(True, alpha=0.3)
plt.show()

# Plot condensed tree
plt.figure(figsize=(10, 6))
clusterer.condensed_tree_.plot(
    select_clusters=True,
    selection_palette=sns.color_palette("husl", 8)
)
plt.title('HDBSCAN Condensed Tree (White Square - Red Channel)')
plt.show()
```

```
# Basic cluster stats
print("\nCluster Membership:")
print(pd.Series(clusterer.labels_).value_counts())
print(f"\nNoise Points: {(clusterer.labels_ == -1).mean()*100:.2f}%")
```

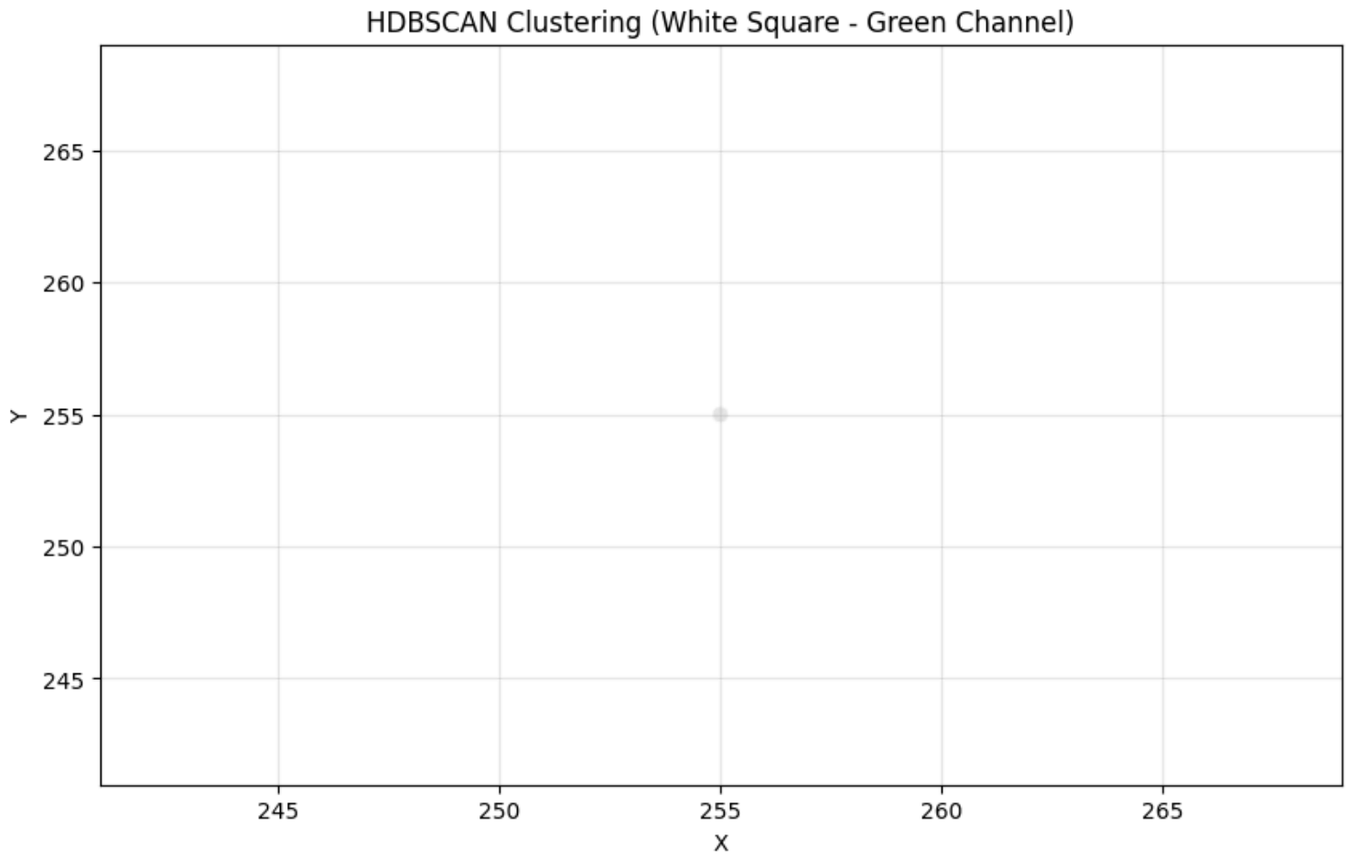

HDBSCAN Clustering (White Square - Red Channel)
265
260
255
250
245
Y
245
250
255
260
265
X

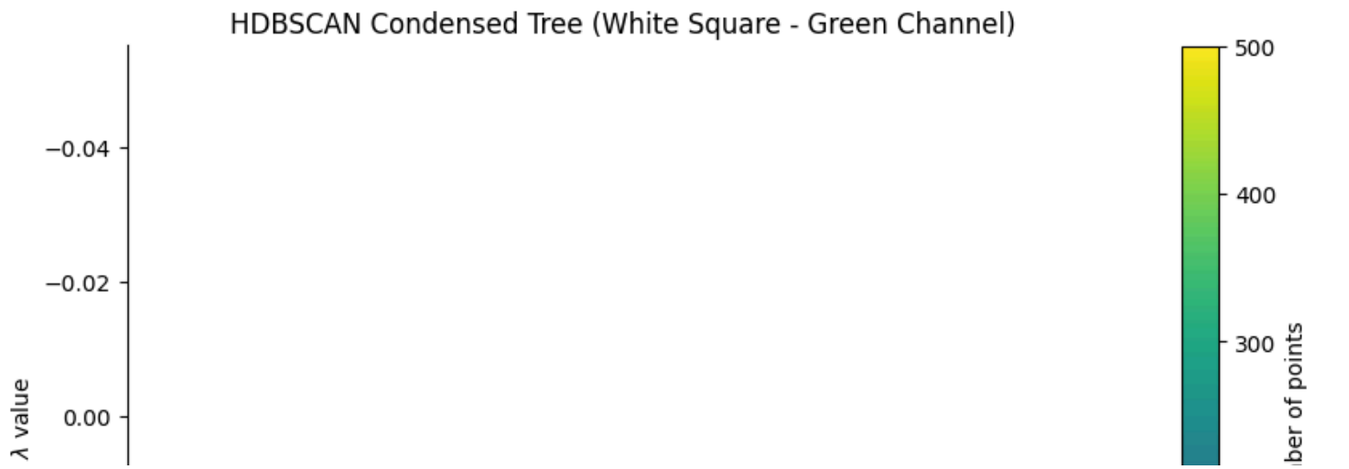

HDBSCAN Condensed Tree (White Square - Red Channel)
−0.04
−0.02
0.00
λ value
500
400
300
ber of points


```
Cluster Membership:
-1    500
Name: count, dtype: int64

Noise Points: 100.00%
```

```
import hdbscan
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

import warnings
warnings.filterwarnings("ignore")

# Load the data
projection = np.loadtxt("/content/whitesquareG.csv", delimiter=",")

# Run HDBSCAN clustering
clusterer = hdbscan.HDBSCAN(
    min_cluster_size=20,
    min_samples=5,
    cluster_selection_method='eom',
    gen_min_span_tree=True
)
clusterer.fit(projection)

# Plot clusters
palette = sns.color_palette("husl", np.unique(clusterer.labels_).max() + 1)
cluster_colors = [palette[col] if col >= 0 else (0.9, 0.9, 0.9) for col in clusterer.labels_]
```

```
plt.figure(figsize=(10, 6))
plt.scatter(
    projection[:, 0], projection[:, 1],
    c=cluster_colors,
    s=50,
    alpha=0.7,
    edgecolor='w',
    linewidth=0.5
)
plt.title('HDBSCAN Clustering (White Square - Green Channel)')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid(True, alpha=0.3)
plt.show()

# Plot condensed tree
plt.figure(figsize=(10, 6))
clusterer.condensed_tree_.plot(
    select_clusters=True,
    selection_palette=sns.color_palette("husl", 8)
)
plt.title('HDBSCAN Condensed Tree (White Square - Green Channel)')
plt.show()

# Basic cluster stats
print("\nCluster Membership:")
print(pd.Series(clusterer.labels_).value_counts())
print(f"\nNoise Points: {(clusterer.labels_ == -1).mean()*100:.2f}%")
```

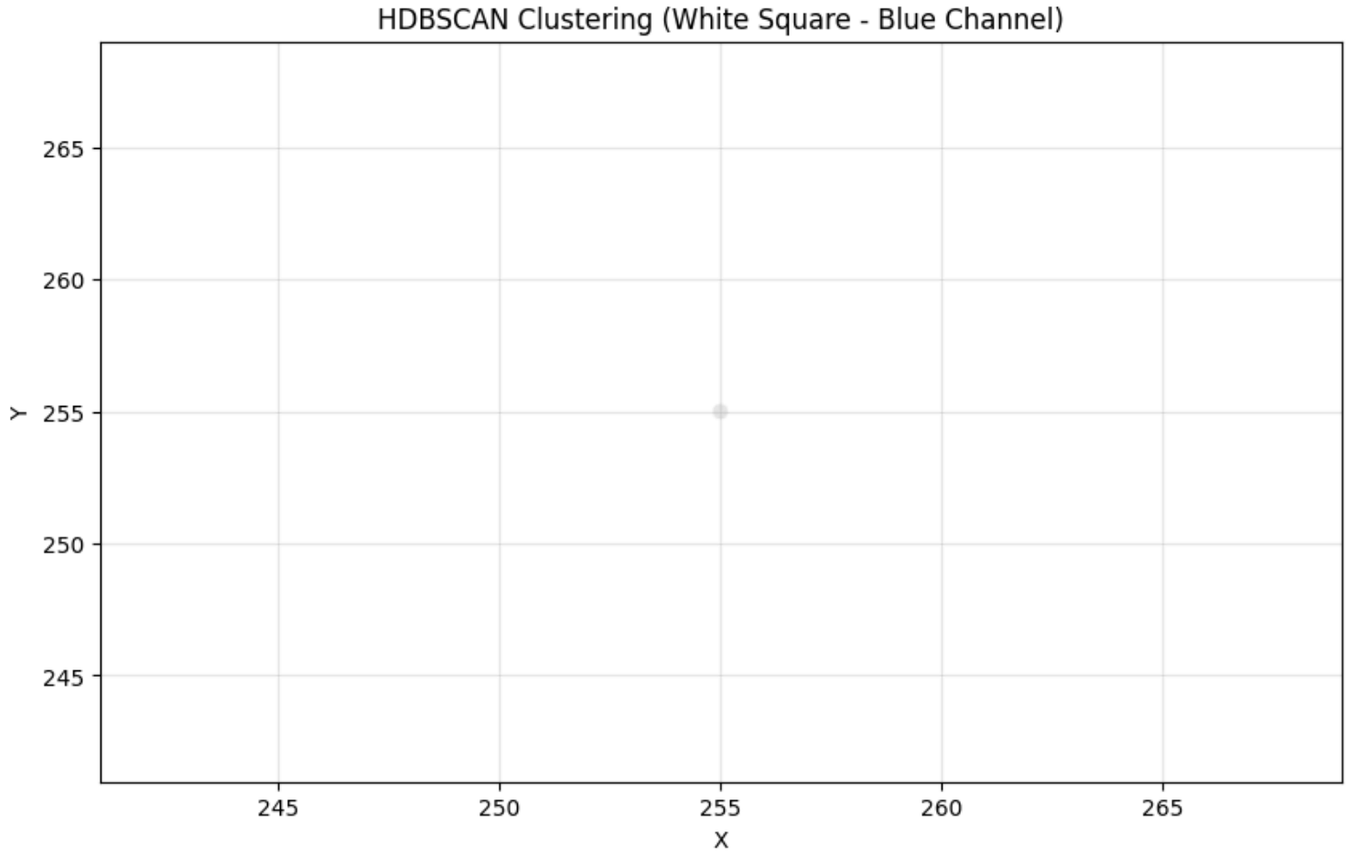

HDBSCAN Clustering (White Square - Green Channel)
265
260
255
250
245
Y
245
250
255
260
265
X

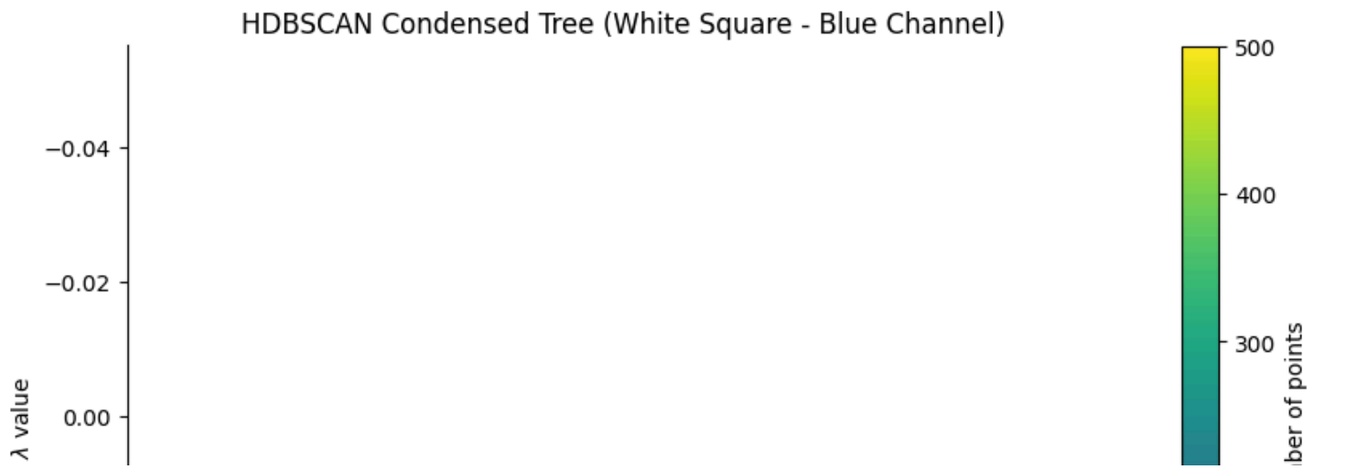

HDBSCAN Condensed Tree (White Square - Green Channel)
−0.04
−0.02
0.00
λ value
500
400
300
ber of points


```
Cluster Membership:
-1    500
Name: count, dtype: int64

Noise Points: 100.00%
```

```
import hdbscan
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

import warnings
warnings.filterwarnings("ignore")

# Load the data
projection = np.loadtxt("/content/whitesquareB.csv", delimiter=",")

# Run HDBSCAN clustering
clusterer = hdbscan.HDBSCAN(
    min_cluster_size=20,
    min_samples=5,
    cluster_selection_method='eom',
    gen_min_span_tree=True
)
clusterer.fit(projection)

# Plot clusters
palette = sns.color_palette("husl", np.unique(clusterer.labels_).max() + 1)
cluster_colors = [palette[col] if col >= 0 else (0.9, 0.9, 0.9) for col in clusterer.labels_]
```

```
plt.figure(figsize=(10, 6))
plt.scatter(
    projection[:, 0], projection[:, 1],
    c=cluster_colors,
    s=50,
    alpha=0.7,
    edgecolor='w',
    linewidth=0.5
)
plt.title('HDBSCAN Clustering (White Square - Blue Channel)')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid(True, alpha=0.3)
plt.show()

# Plot condensed tree
plt.figure(figsize=(10, 6))
clusterer.condensed_tree_.plot(
    select_clusters=True,
    selection_palette=sns.color_palette("husl", 8)
)
plt.title('HDBSCAN Condensed Tree (White Square - Blue Channel)')
plt.show()

# Basic cluster stats
print("\nCluster Membership:")
print(pd.Series(clusterer.labels_).value_counts())
print(f"\nNoise Points: {(clusterer.labels_ == -1).mean()*100:.2f}%")
```


HDBSCAN Clustering (White Square - Blue Channel)
265
260
255
250
245
Y
245
250
255
260
265
X


HDBSCAN Condensed Tree (White Square - Blue Channel)
−0.04
−0.02
0.00
λ value
500
400
300
ber of points

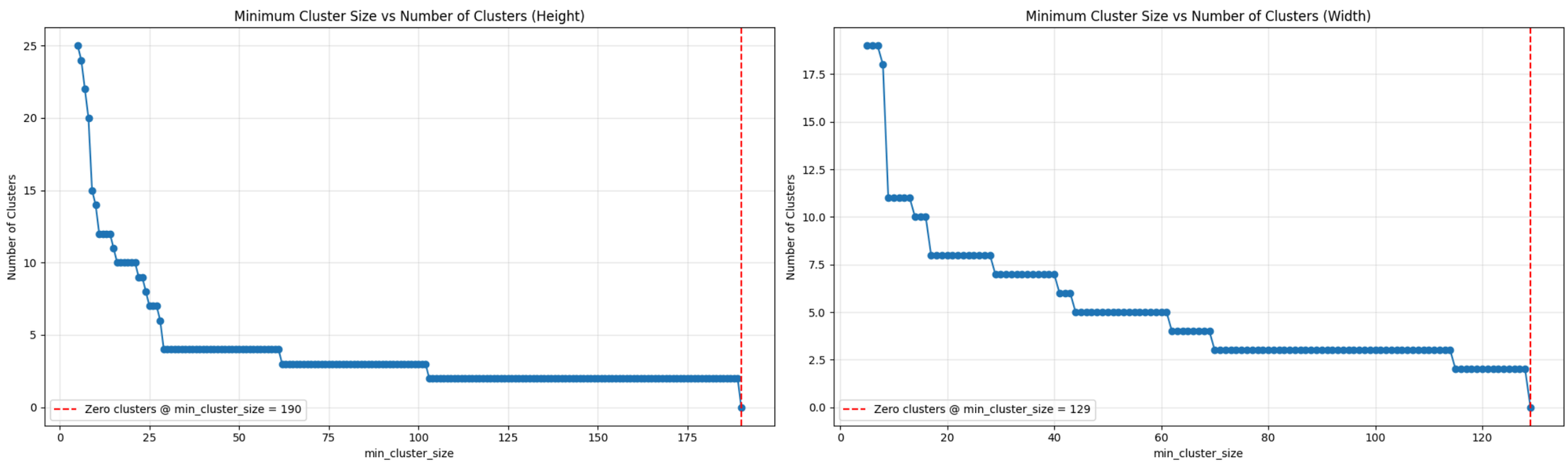

```
Cluster Membership:
-1    500
Name: count, dtype: int64

Noise Points: 100.00%
```

Attempt to Deconvolute Columbia Lion

```
import hdbscan
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

# Load the data
projection = np.loadtxt("/content/Lion_B - Lion_B.csv", delimiter=",")
projection_t = projection.T

# Create figure with two subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6))

# First plot (Height)
# Initialize tracking lists
min_cluster_sizes_height = []
n_clusters_height = []
# Start sweep
size = 5
while True:
    clusterer = hdbscan.HDBSCAN(
        min_cluster_size=size,
```

```
        min_samples=5,
        cluster_selection_method='eom'
    )
    clusterer.fit(projection)
    labels = clusterer.labels_
    num_clusters = len(set(labels) - {-1})

    min_cluster_sizes_height.append(size)
    n_clusters_height.append(num_clusters)

    if num_clusters == 0:
        break
    size += 1

ax1.plot(min_cluster_sizes_height, n_clusters_height, marker='o', linestyle='-')
ax1.axvline(size, color='red', linestyle='--', label=f"Zero clusters @ min_cluster_size = {size}")
ax1.set_title('Minimum Cluster Size vs Number of Clusters (Height)')
ax1.set_xlabel('min_cluster_size')
ax1.set_ylabel('Number of Clusters')
ax1.legend()
ax1.grid(True, alpha=0.3)

# Second plot (Width)
# Initialize tracking lists
min_cluster_sizes_width = []
n_clusters_width = []
# Start sweep
size = 5
while True:
    clusterer = hdbscan.HDBSCAN(
        min_cluster_size=size,
        min_samples=5,
        cluster_selection_method='eom'
    )
    clusterer.fit(projection_t)
    labels = clusterer.labels_
    num_clusters = len(set(labels) - {-1})

    min_cluster_sizes_width.append(size)
    n_clusters_width.append(num_clusters)

    if num_clusters == 0:
        break
    size += 1
```

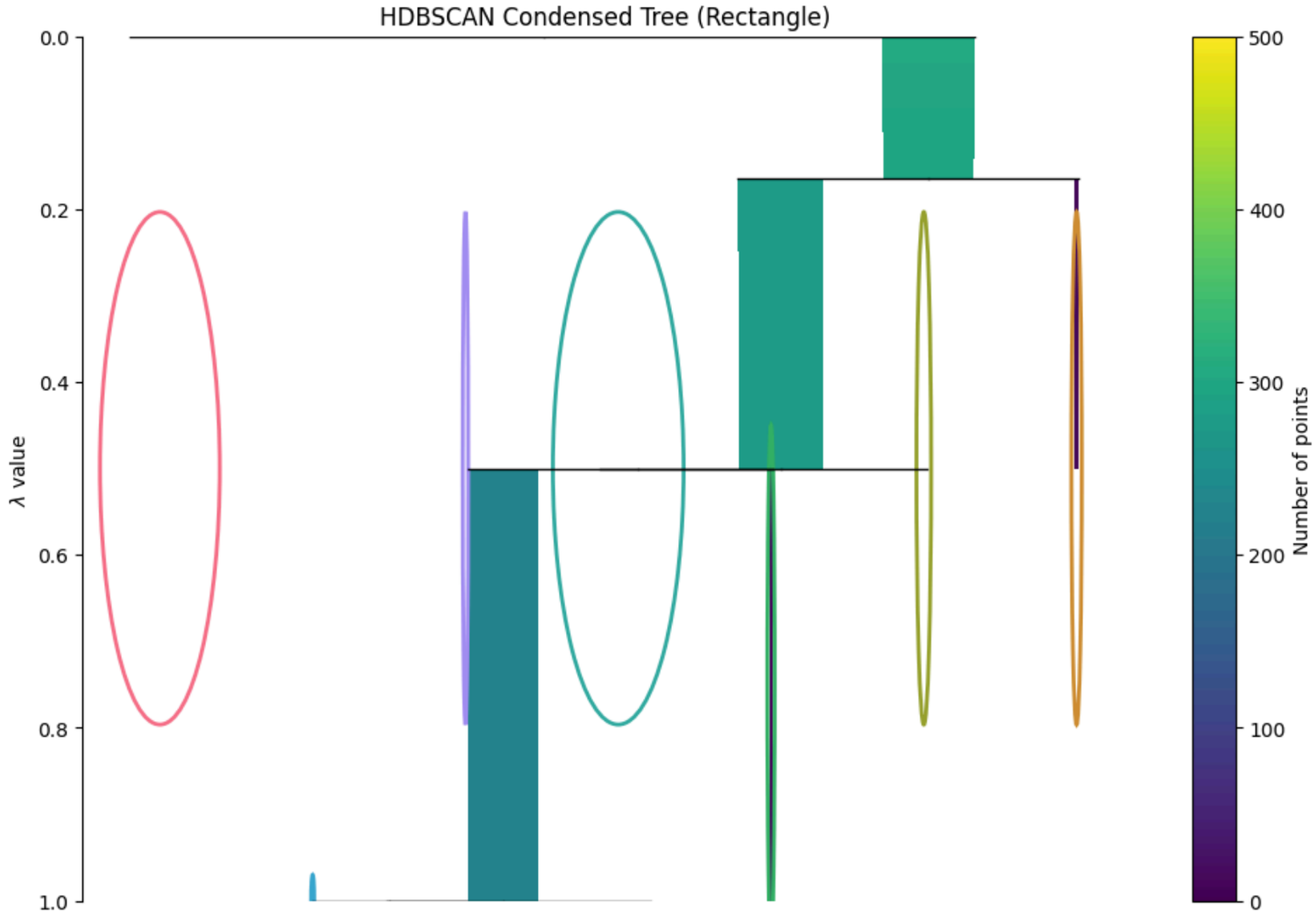
```
ax2.plot(min_cluster_sizes_width, n_clusters_width, marker='o', linestyle='-')
ax2.axvline(size, color='red', linestyle='--', label=f"Zero clusters @ min_cluster_size = {size}")
ax2.set_title('Minimum Cluster Size vs Number of Clusters (Width)')
ax2.set_xlabel('min_cluster_size')
ax2.set_ylabel('Number of Clusters')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```



Rectangle

```
import hdbscan
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
import pandas as pd

import warnings
warnings.filterwarnings("ignore")

# Load the data
projection = np.loadtxt("/content/BlackRectangle_B - BlackRectangle_B.csv",
                        delimiter=",")
projection_t = projection.T

# Run HDBSCAN clustering
clusterer = hdbscan.HDBSCAN(
    min_cluster_size=5,
    min_samples=5,
    cluster_selection_method='eom',
    gen_min_span_tree=True
)
clusterer.fit(projection_t)

# Plot clusters
palette = sns.color_palette("husl", np.unique(clusterer.labels_).max() + 1)
cluster_colors = [palette[col] if col >= 0 else (0.9, 0.9, 0.9) for col in clusterer.labels_]

# Plot condensed tree
plt.figure(figsize=(12, 8))
clusterer.condensed_tree_.plot(
    select_clusters=True,
    selection_palette=sns.color_palette("husl", 8)
)
plt.title('HDBSCAN Condensed Tree (Rectangle)')
plt.show()

# Basic cluster stats
print("\nCluster Membership:")
print(pd.Series(clusterer.labels_).value_counts())
print(f"\nNoise Points: {(clusterer.labels_ == -1).mean()*100:.2f}%")
```

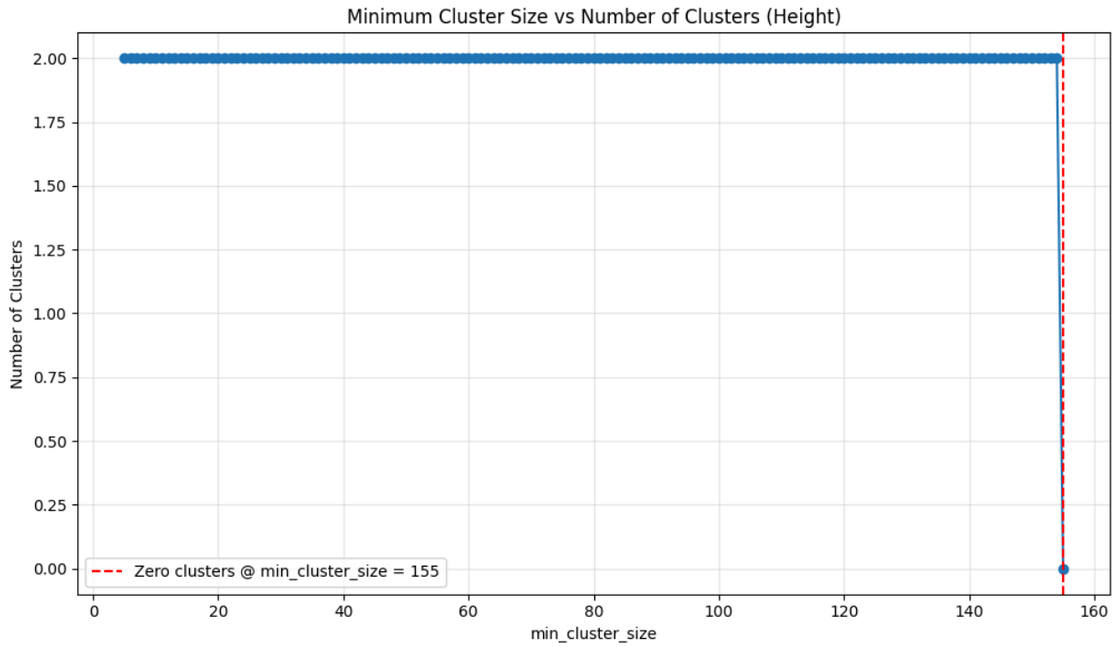


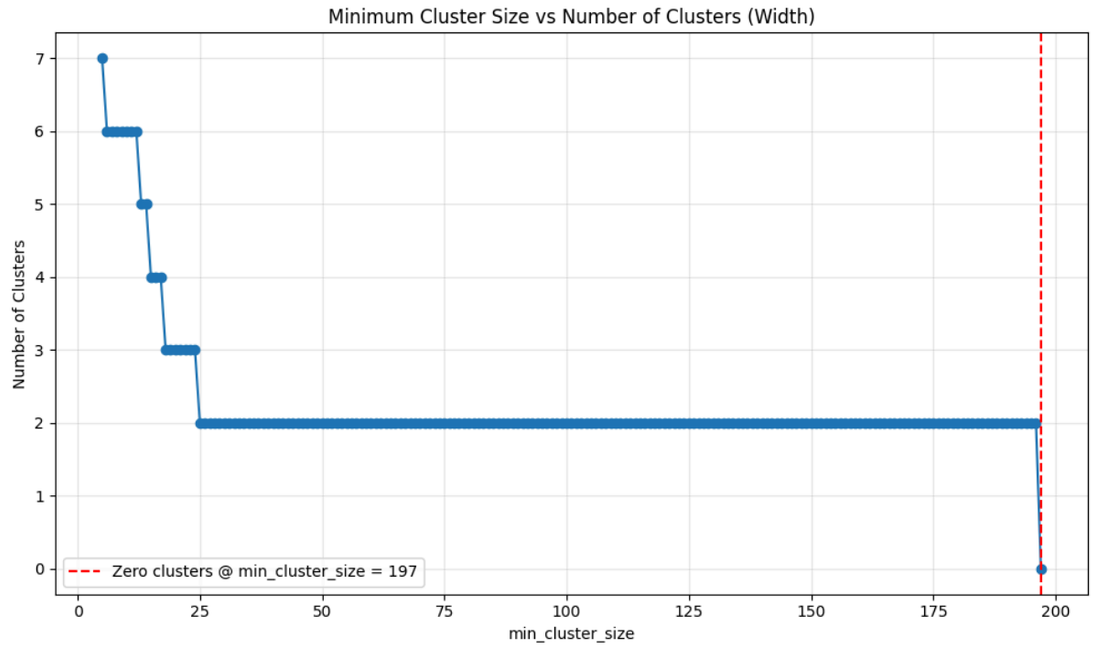

```
Cluster Membership:
 4    214
 0    196
 2     24
-1     23
 1     17
 3     12
 6      9
 5      5
```

```
Name: count, dtype: int64

Noise Points: 4.60%
```

```
import hdbscan
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

# Load the data
projection = np.loadtxt("/content/BlackRectangle_B - BlackRectangle_B.csv", delimiter=",")
projection_t = projection.T

# Create figure with two subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6))

# First plot (Height)
# Initialize tracking lists
min_cluster_sizes_height = []
n_clusters_height = []
# Start sweep
size = 5
while True:
    clusterer = hdbscan.HDBSCAN(
        min_cluster_size=size,
        min_samples=5,
        cluster_selection_method='eom'
    )
    clusterer.fit(projection)
    labels = clusterer.labels_
    num_clusters = len(set(labels) - {-1})

    min_cluster_sizes_height.append(size)
    n_clusters_height.append(num_clusters)

    if num_clusters == 0:
        break
    size += 1

ax1.plot(min_cluster_sizes_height, n_clusters_height, marker='o', linestyle='-')
ax1.axvline(size, color='red', linestyle='--', label=f"Zero clusters @ min_cluster_size = {size}")
ax1.set_title('Minimum Cluster Size vs Number of Clusters (Height)')
```

```
ax1.set_xlabel('min_cluster_size')
ax1.set_ylabel('Number of Clusters')
ax1.legend()
ax1.grid(True, alpha=0.3)

# Second plot (Width)
# Initialize tracking lists
min_cluster_sizes_width = []
n_clusters_width = []
# Start sweep
size = 5
while True:
    clusterer = hdbscan.HDBSCAN(
        min_cluster_size=size,
        min_samples=5,
        cluster_selection_method='eom'
    )
    clusterer.fit(projection_t)
    labels = clusterer.labels_
    num_clusters = len(set(labels) - {-1})

    min_cluster_sizes_width.append(size)
    n_clusters_width.append(num_clusters)

    if num_clusters == 0:
        break
    size += 1

ax2.plot(min_cluster_sizes_width, n_clusters_width, marker='o', linestyle='-')
ax2.axvline(size, color='red', linestyle='--', label=f"Zero clusters @ min_cluster_size = {size}")
ax2.set_title('Minimum Cluster Size vs Number of Clusters (Width)')
ax2.set_xlabel('min_cluster_size')
ax2.set_ylabel('Number of Clusters')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```





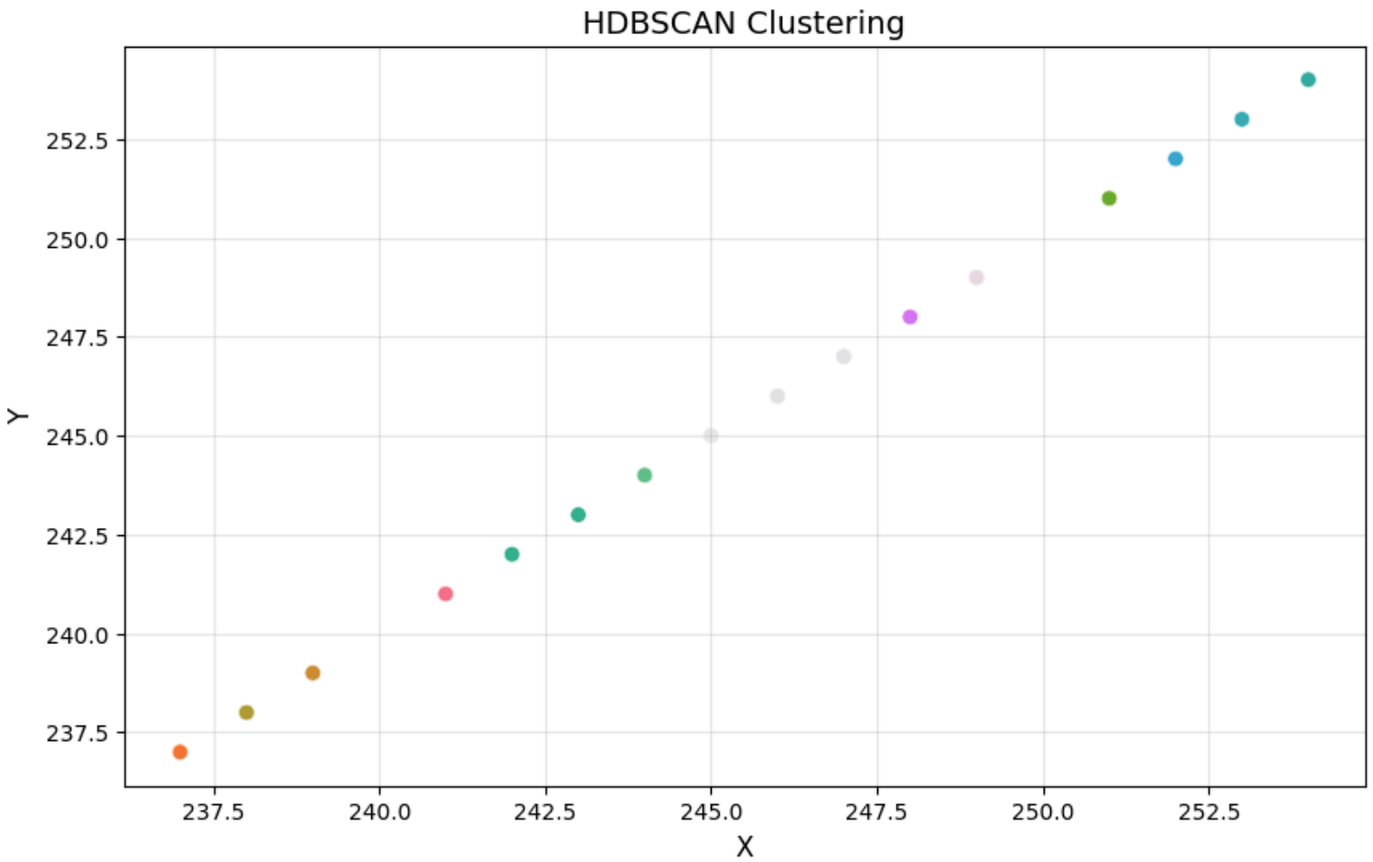
Use Case: Blurry Images

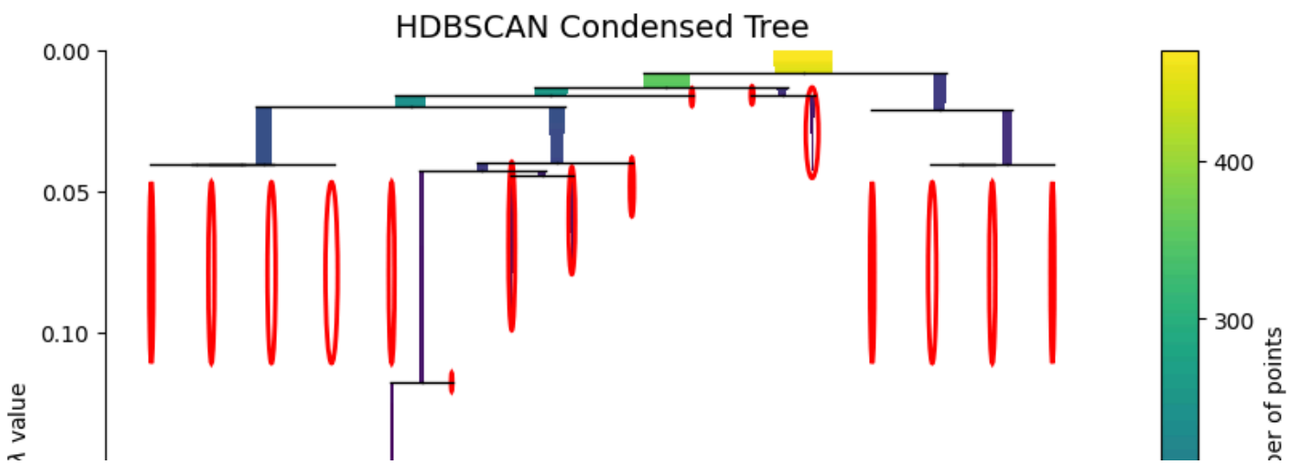
Semi-Circle 1

```
import hdbscan
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

import warnings
warnings.filterwarnings("ignore")

# --- Load Data ---
projection = np.loadtxt("/content/blurry_semi_B - blurry_semi_B.csv", delimiter=",")
```

```
# --- Clustering ---
clusterer = hdbscan.HDBSCAN(
    min_cluster_size=5,
    min_samples=5,
    cluster_selection_method='eom',
    gen_min_span_tree=True
)
clusterer.fit(projection)

# --- Plot Clusters ---
palette = sns.color_palette("husl", np.unique(clusterer.labels_).max() + 1)
cluster_colors = [
    palette[col] if col >= 0 else (0.9, 0.9, 0.9)
    for col in clusterer.labels_
]

plt.figure(figsize=(10, 6))
plt.scatter(
    projection[:, 0], projection[:, 1],
    c=cluster_colors,
    s=50,
    alpha=0.7,
    edgecolor='w',
    linewidth=0.5
)
plt.title('HDBSCAN Clustering', fontsize=14)
plt.xlabel('X', fontsize=12)
plt.ylabel('Y', fontsize=12)
plt.grid(True, alpha=0.3)
plt.show()

# --- Condensed Tree (for Stability Analysis) ---
plt.figure(figsize=(10, 6))
clusterer.condensed_tree_.plot(
    select_clusters=True,
    selection_palette=sns.color_palette("husl", 8),
)
plt.title('HDBSCAN Condensed Tree', fontsize=14)
plt.show()

# --- Additional Diagnostics ---
# Print cluster statistics
cluster_stats = pd.Series(clusterer.labels_).value_counts()
print("\nCluster Membership:")
```

```
print(cluster_stats)

# Check % of noise points
noise_ratio = (clusterer.labels_ == -1).mean() * 100
print(f"\nNoise Points: {noise_ratio:.2f}%")
```

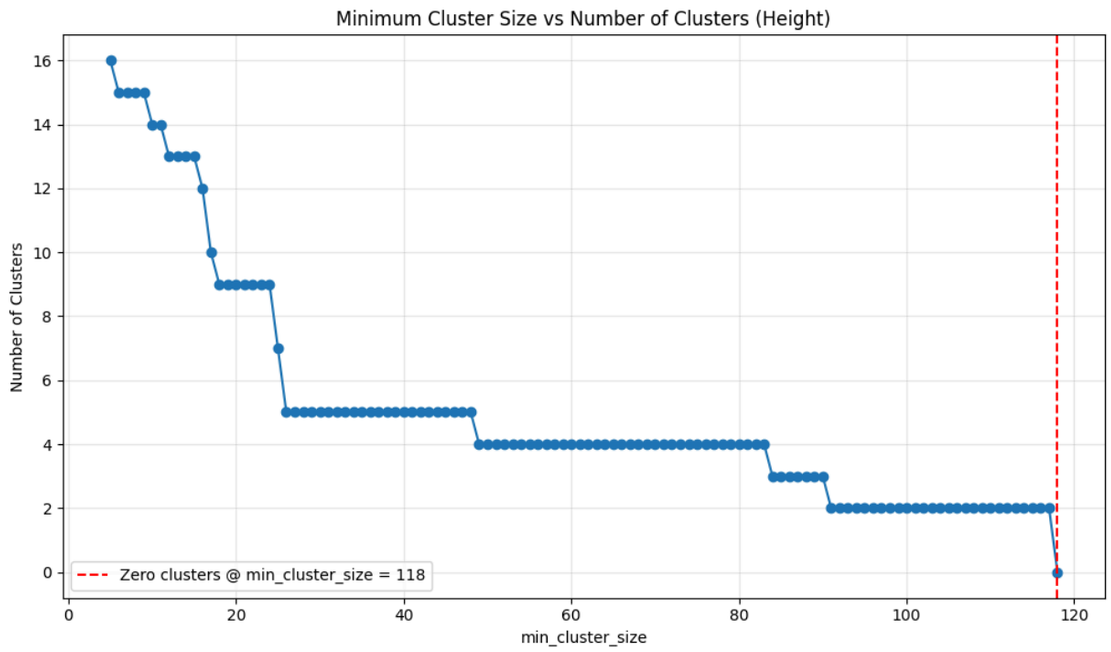

HDBSCAN Clustering
252.5
250.0
247.5
245.0
242.5
240.0
237.5
Y
237.5
240.0
242.5
245.0
247.5
250.0
252.5
X

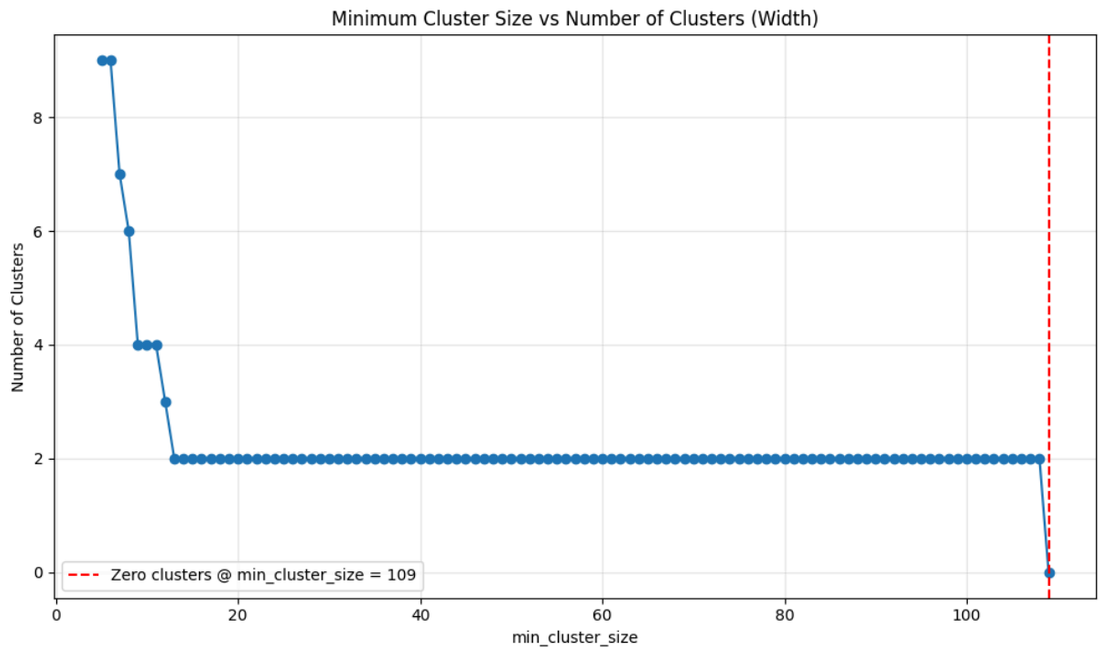

HDBSCAN Condensed Tree
0.00
0.05
0.10
λ value
400
300
er of points


```
Cluster Membership:
-1     97
 7     50
 5     48
 2     32
 10    32
 13    26
 12    26
 3     24
 9     24
 14    23
 0     17
 8     16
 11    16
 1     15
 6     11
 4      9
 15     5
Name: count, dtype: int64

Noise Points: 20.59%
```

```
import hdbscan
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

# Load the data
projection = np.loadtxt("/content/blurry_semi_B - blurry_semi_B.csv", delimiter=",")
projection_t = projection.T

# Create figure with two subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6))

# First plot (Height)
# Initialize tracking lists
min_cluster_sizes_height = []
n_clusters_height = []
# Start sweep
size = 5
while True:
    clusterer = hdbscan.HDBSCAN(
        min_cluster_size=size,
        min_samples=5,
        cluster_selection_method='eom'
    )
    clusterer.fit(projection)
    labels = clusterer.labels_
    num_clusters = len(set(labels) - {-1})

    min_cluster_sizes_height.append(size)
    n_clusters_height.append(num_clusters)

    if num_clusters == 0:
        break
    size += 1

ax1.plot(min_cluster_sizes_height, n_clusters_height, marker='o', linestyle='-')
ax1.axvline(size, color='red', linestyle='--', label=f"Zero clusters @ min_cluster_size = {size}")
ax1.set_title('Minimum Cluster Size vs Number of Clusters (Height)')
ax1.set_xlabel('min_cluster_size')
ax1.set_ylabel('Number of Clusters')
ax1.legend()
ax1.grid(True, alpha=0.3)
```

```
# Second plot (Width)
# Initialize tracking lists
min_cluster_sizes_width = []
n_clusters_width = []
# Start sweep
size = 5
while True:
    clusterer = hdbscan.HDBSCAN(
        min_cluster_size=size,
        min_samples=5,
        cluster_selection_method='eom'
    )
    clusterer.fit(projection_t)
    labels = clusterer.labels_
    num_clusters = len(set(labels) - {-1})

    min_cluster_sizes_width.append(size)
    n_clusters_width.append(num_clusters)

    if num_clusters == 0:
        break
    size += 1

ax2.plot(min_cluster_sizes_width, n_clusters_width, marker='o', linestyle='-')
ax2.axvline(size, color='red', linestyle='--', label=f"Zero clusters @ min_cluster_size = {size}")
ax2.set_title('Minimum Cluster Size vs Number of Clusters (Width)')
ax2.set_xlabel('min_cluster_size')
ax2.set_ylabel('Number of Clusters')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```



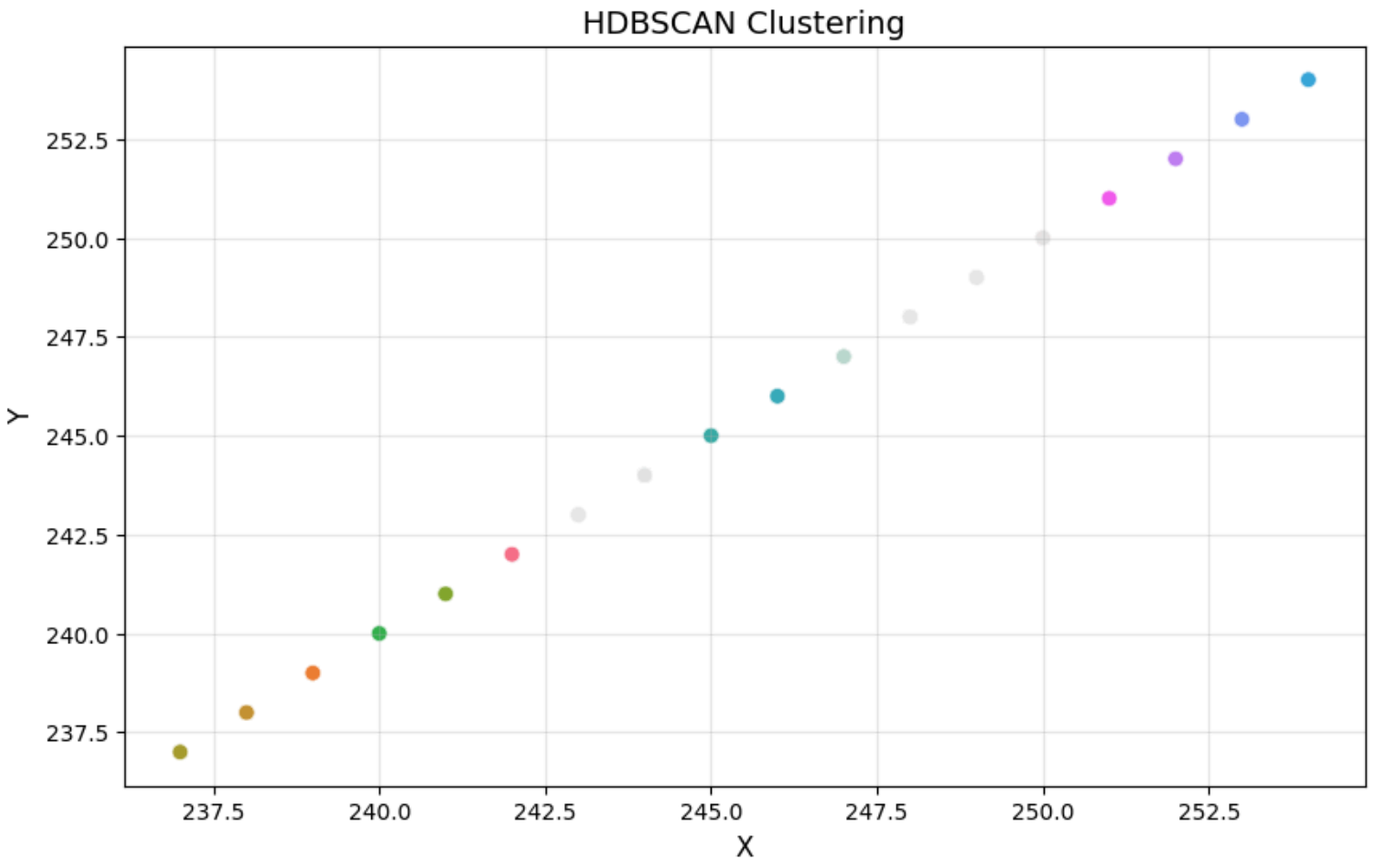


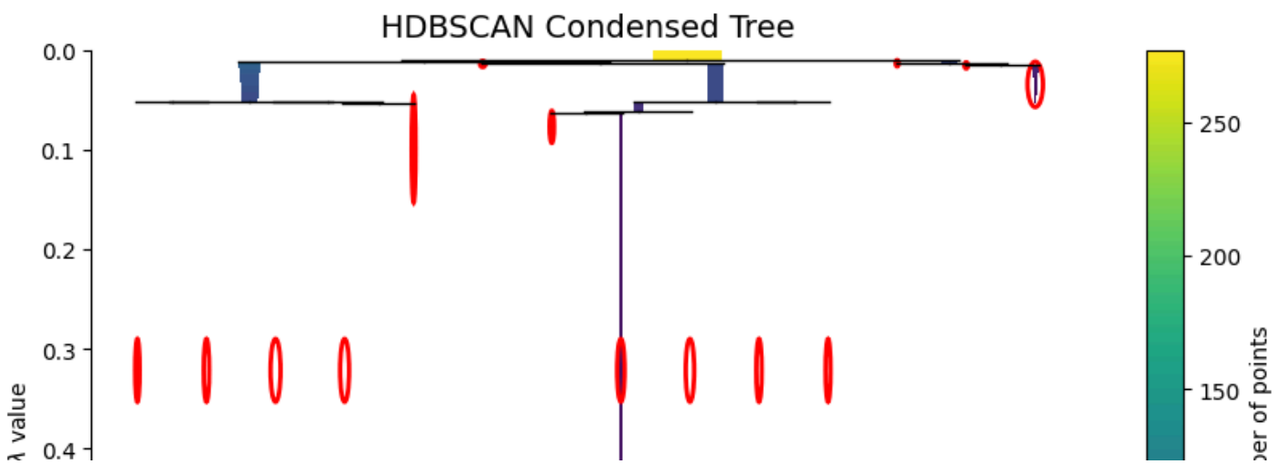
Semi-Circle 2

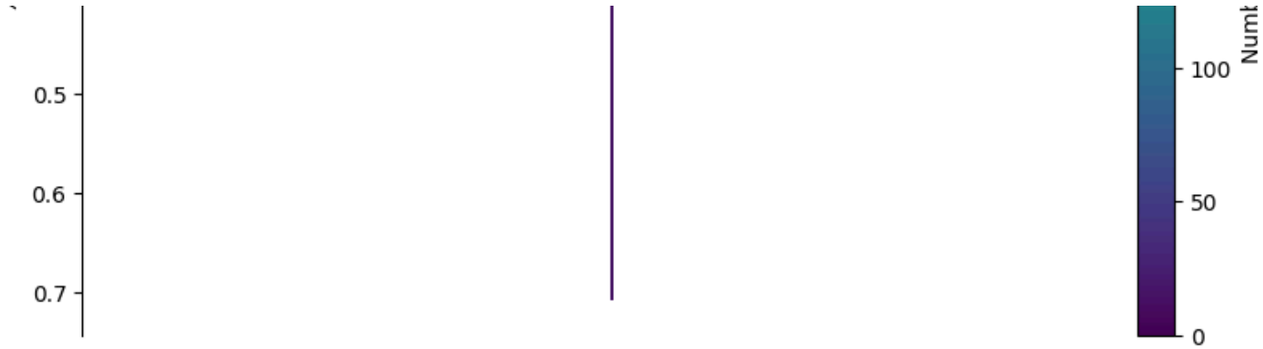
```
import hdbscan
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

import warnings
warnings.filterwarnings("ignore")

# --- Load Data ---
projection = np.loadtxt("/content/blurred_semi_2_B - blurred_semi_2_B.csv", delimiter=",")

# --- Clustering ---
clusterer = hdbscan.HDBSCAN(
    min_cluster_size=5,
    min_samples=5,
```

```
    cluster_selection_method='eom',
    gen_min_span_tree=True
)
clusterer.fit(projection)

# --- Plot Clusters ---
palette = sns.color_palette("husl", np.unique(clusterer.labels_).max() + 1)
cluster_colors = [
    palette[col] if col >= 0 else (0.9, 0.9, 0.9)
    for col in clusterer.labels_
]

plt.figure(figsize=(10, 6))
plt.scatter(
    projection[:, 0], projection[:, 1],
    c=cluster_colors,
    s=50,
    alpha=0.7,
    edgecolor='w',
    linewidth=0.5
)
plt.title('HDBSCAN Clustering', fontsize=14)
plt.xlabel('X', fontsize=12)
plt.ylabel('Y', fontsize=12)
plt.grid(True, alpha=0.3)
plt.show()

# --- Condensed Tree (for Stability Analysis) ---
plt.figure(figsize=(10, 6))
clusterer.condensed_tree_.plot(
    select_clusters=True,
    selection_palette=sns.color_palette("husl", 8),
)
plt.title('HDBSCAN Condensed Tree', fontsize=14)
plt.show()

# --- Additional Diagnostics ---
# Print cluster statistics
cluster_stats = pd.Series(clusterer.labels_).value_counts()
print("\nCluster Membership:")
print(cluster_stats)

# Check % of noise points
```

```
noise_ratio = (clusterer.labels_ == -1).mean() * 100
print(f"\nNoise Points: {noise_ratio:.2f}%")
```


HDBSCAN Clustering
252.5
250.0
247.5
245.0
242.5
240.0
237.5
Y
237.5
240.0
242.5
245.0
247.5
250.0
252.5
X


HDBSCAN Condensed Tree
0.0
0.1
0.2
0.3
0.4
λ value
250
200
150
er of points

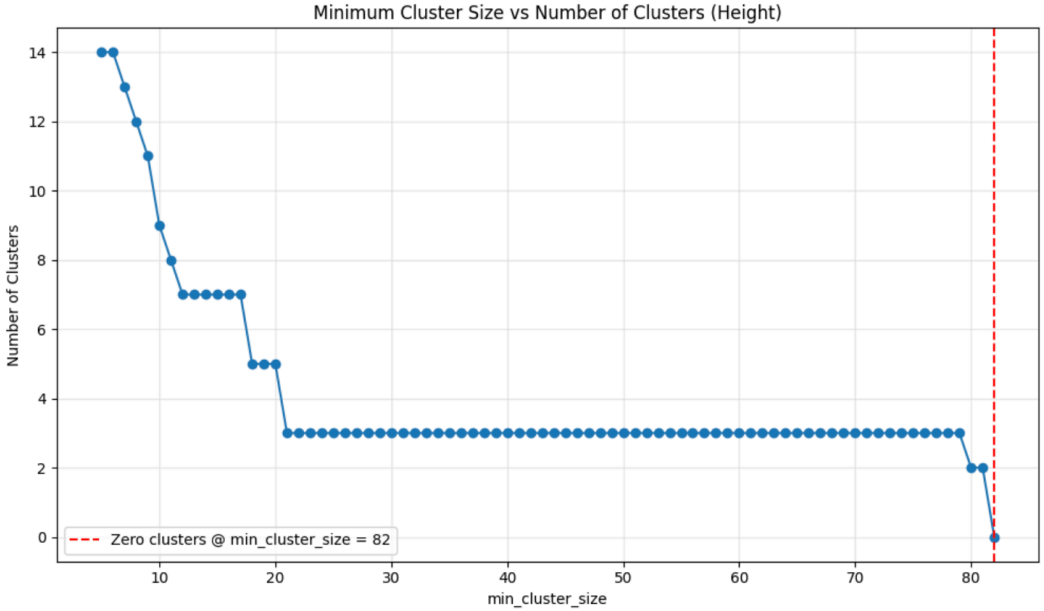


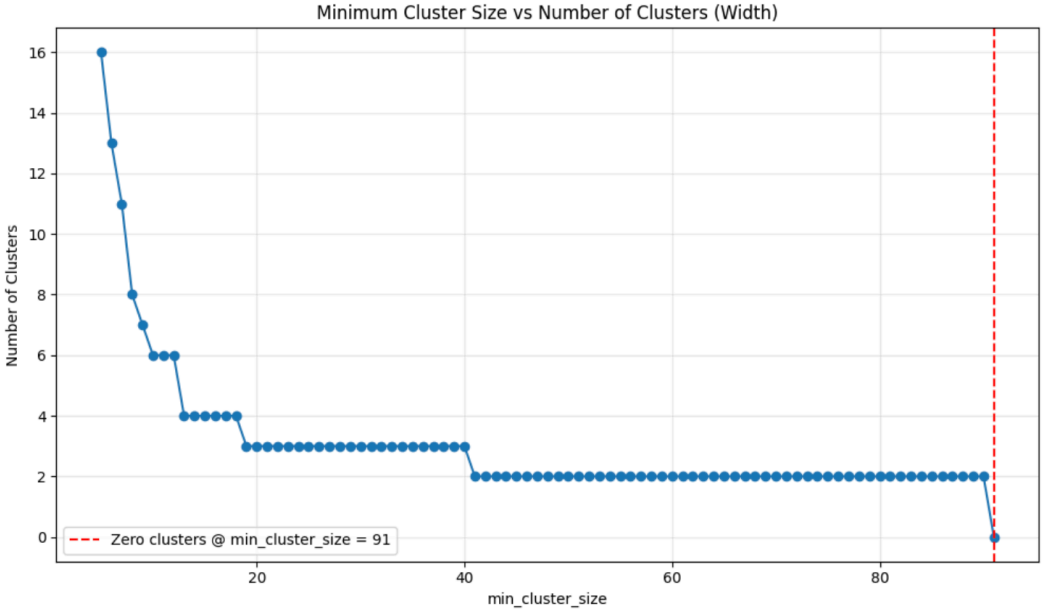

```
Cluster Membership:
-1     94
 8     31
 11    20
 12    18
 1     17
 5     15
 10    11
 2     11
 0     11
 3     10
 9      9
 4      9
 13     8
 7      7
 6      6
Name: count, dtype: int64
```

Noise Points: 33.94%

```
import hdbscan
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

# Load the data
projection = np.loadtxt("/content/blurred_semi_2_B - blurred_semi_2_B.csv",
                        delimiter=",")
projection_t = projection.T

# Create figure with two subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6))

# First plot (Height)
# Initialize tracking lists
min_cluster_sizes_height = []
n_clusters_height = []
# Start sweep
size = 5
while True:
    clusterer = hdbscan.HDBSCAN(
        min_cluster_size=size,
        min_samples=5,
        cluster_selection_method='eom'
    )
    clusterer.fit(projection)
    labels = clusterer.labels_
    num_clusters = len(set(labels) - {-1})

    min_cluster_sizes_height.append(size)
    n_clusters_height.append(num_clusters)
```

```
    if num_clusters == 0:
        break
    size += 1

ax2.plot(min_cluster_sizes_width, n_clusters_width, marker='o', linestyle='-')
ax2.axvline(size, color='red', linestyle='--', label=f"Zero clusters @ min_cluster_size = {size}")
ax2.set_title('Minimum Cluster Size vs Number of Clusters (Width)')
ax2.set_xlabel('min_cluster_size')
ax2.set_ylabel('Number of Clusters')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

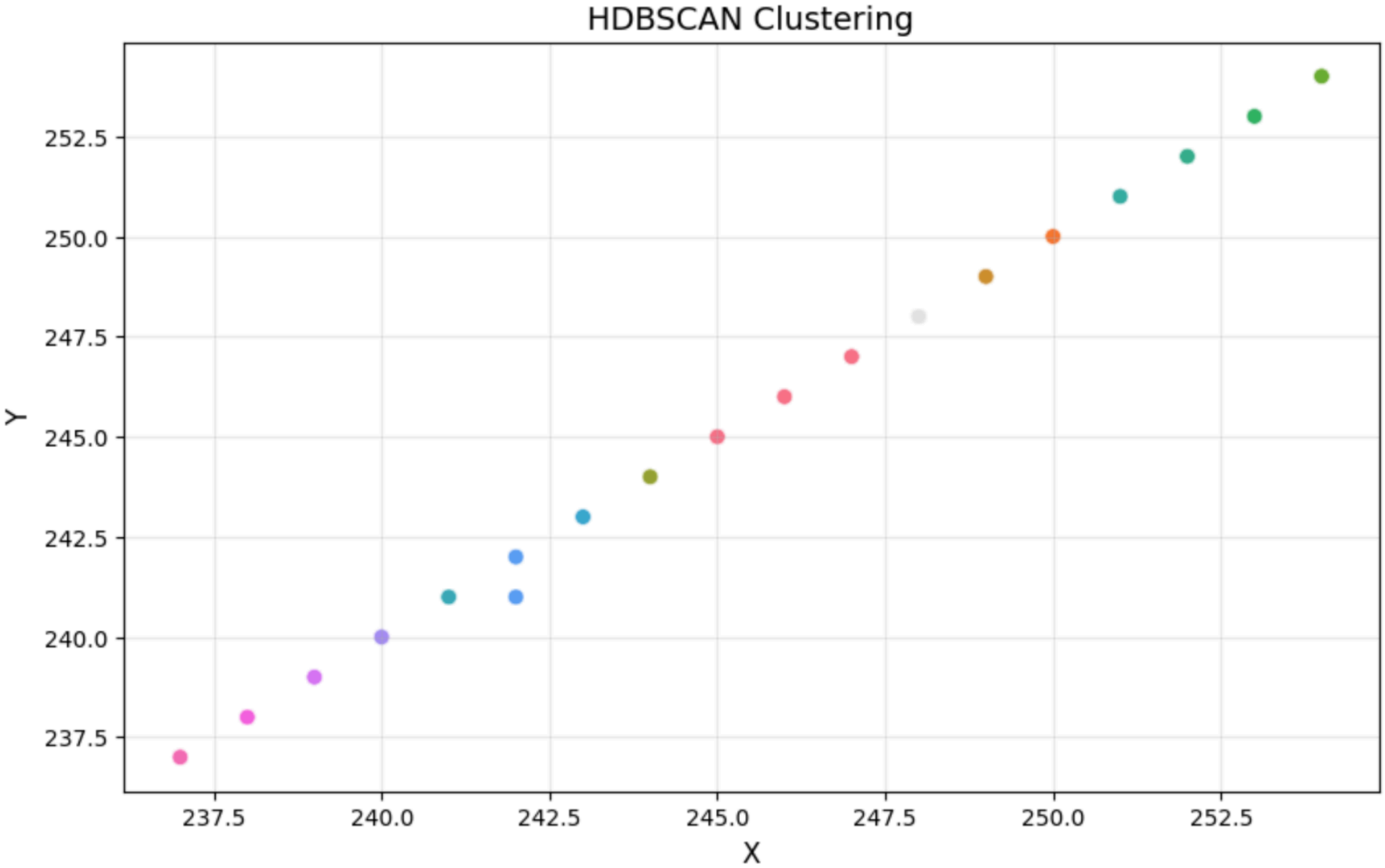

Minimum Cluster Size vs Number of Clusters (Height)
Number of Clusters
min_cluster_size
Zero clusters @ min_cluster_size = 82
0
2
4
6
8
10
12
14
10
20
30
40
50
60
70
80

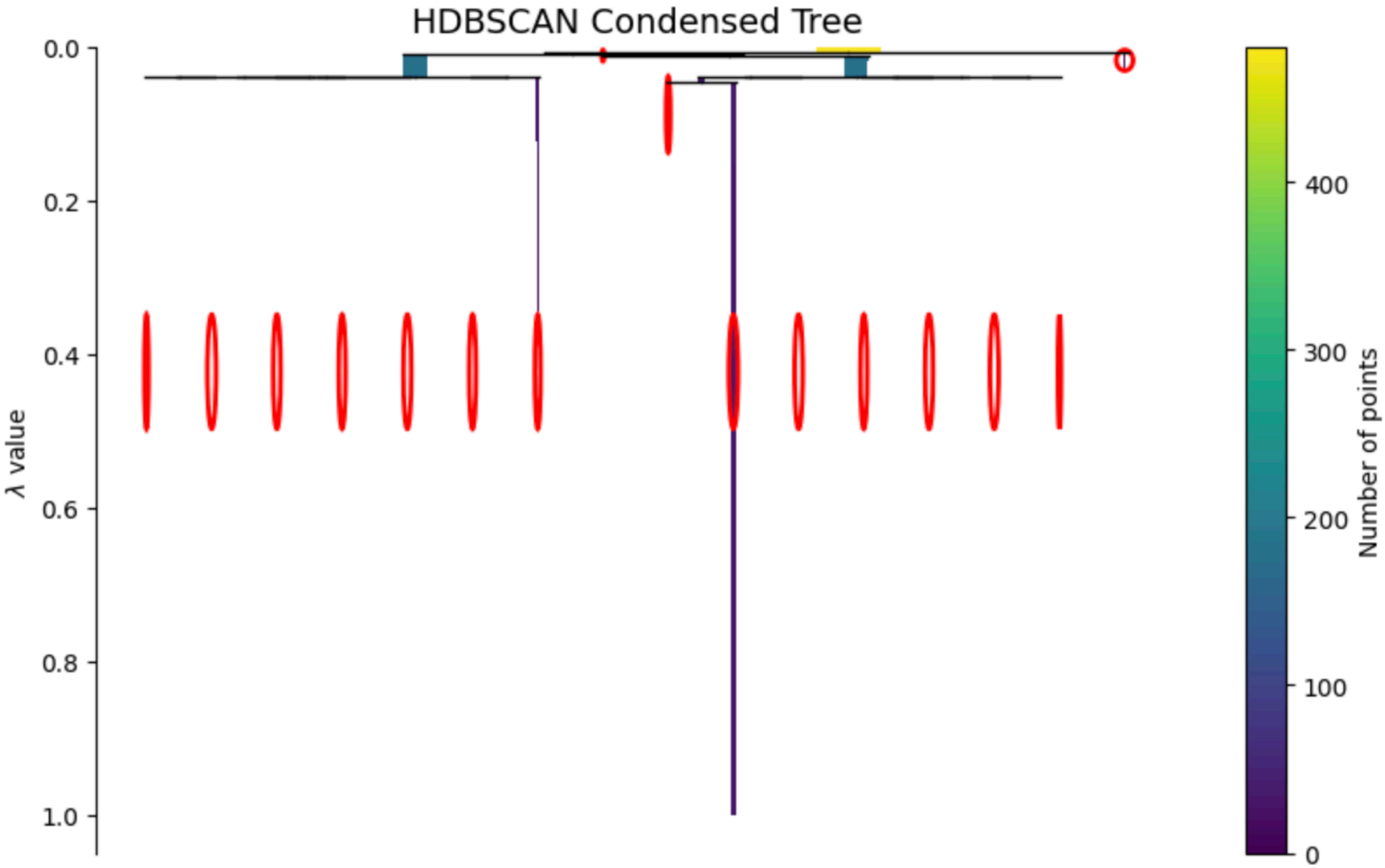

Minimum Cluster Size vs Number of Clusters (Width)
Number of Clusters
min_cluster_size
Zero clusters @ min_cluster_size = 91
0
2
4
6
8
10
12
14
16
20
40
60
80

Oval

```
[ ]  import hdbscan
     import numpy as np
     import seaborn as sns
     import matplotlib.pyplot as plt
     import pandas as pd

     import warnings
     warnings.filterwarnings("ignore")

     # --- Load Data ---
     projection = np.loadtxt("/content/blurry_oval_3_B - blurry_oval_3_B.csv", delimiter=",")

     # --- Clustering ---
     clusterer = hdbscan.HDBSCAN(
         min_cluster_size=5,
         min_samples=5,
         cluster_selection_method='eom',
         gen_min_span_tree=True
     )
     clusterer.fit(projection)

     # --- Plot Clusters ---
     palette = sns.color_palette("husl", np.unique(clusterer.labels_).max() + 1)
     cluster_colors = [
         palette[col] if col >= 0 else (0.9, 0.9, 0.9)
         for col in clusterer.labels_
     ]

     plt.figure(figsize=(10, 6))
     plt.scatter(
```

```
plt.scatter(
    projection[:, 0], projection[:, 1],
    c=cluster_colors,
    s=50,
    alpha=0.7,
    edgecolor='w',
    linewidth=0.5
)
plt.title('HDBSCAN Clustering', fontsize=14)
plt.xlabel('X', fontsize=12)
plt.ylabel('Y', fontsize=12)
plt.grid(True, alpha=0.3)
plt.show()

# --- Condensed Tree (for Stability Analysis) ---
plt.figure(figsize=(10, 6))
clusterer.condensed_tree_.plot(
    select_clusters=True,
    selection_palette=sns.color_palette("husl", 8),
)
plt.title('HDBSCAN Condensed Tree', fontsize=14)
plt.show()

# --- Additional Diagnostics ---
# Print cluster statistics
cluster_stats = pd.Series(clusterer.labels_).value_counts()
print("\nCluster Membership:")
print(cluster_stats)

# Check % of noise points
noise_ratio = (clusterer.labels_ == -1).mean() * 100
print(f"\nNoise Points: {noise_ratio:.2f}%")
```


HDBSCAN Clustering
252.5
250.0
247.5
245.0
242.5
240.0
237.5
Y
237.5
240.0
242.5
245.0
247.5
250.0
252.5
X

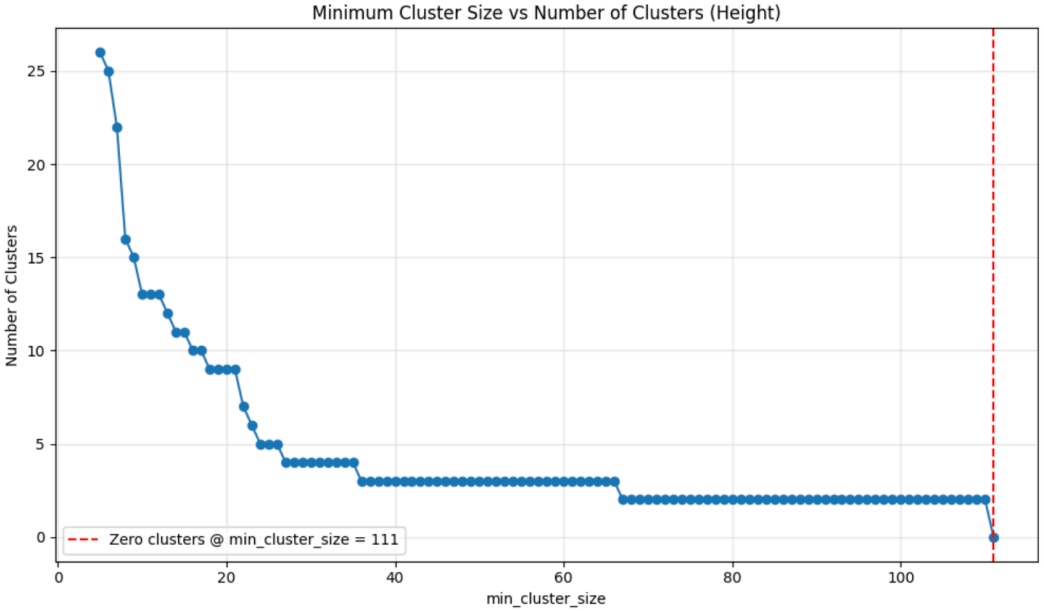

HDBSCAN Condensed Tree
λ value
0.0
0.2
0.4
0.6
0.8
1.0
Number of points
0
100
200
300
400

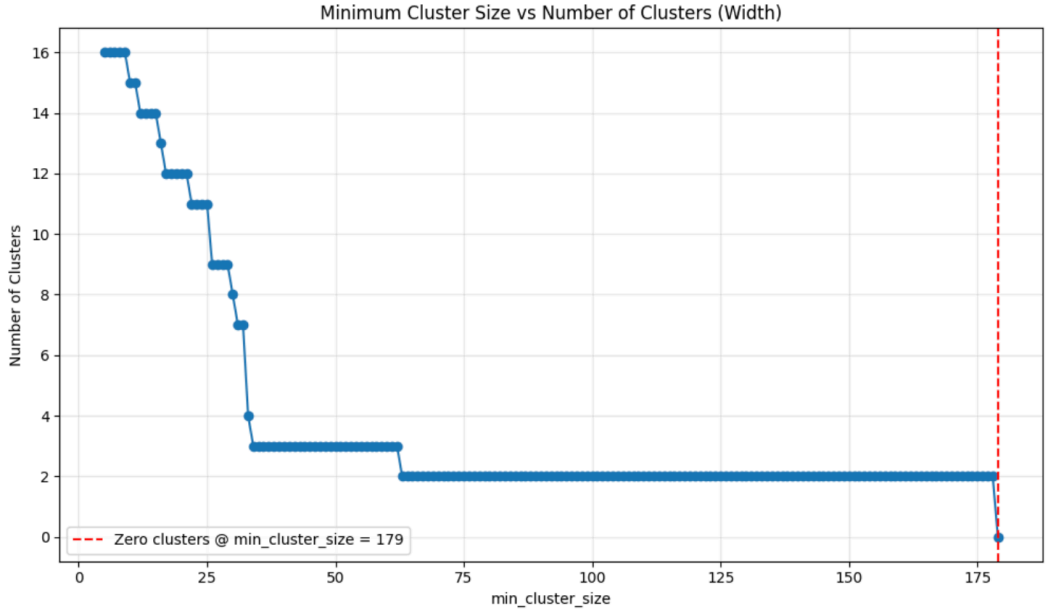
```
Cluster Membership:
 0     62
-1     59
 11    34
 14    32
 6     32
 9     30
 1     29
 13    29
 7     27
 2     25
```

```python
import hdbscan
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")

# Load the data
projection = np.loadtxt("/content/blurry_oval_3_B - blurry_oval_3_B.csv",
                        delimiter=",")
projection_t = projection.T

# Create figure with two subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6))

# First plot (Height)
# Initialize tracking lists
min_cluster_sizes_height = []
n_clusters_height = []
# Start sweep
size = 5
while True:
    clusterer = hdbscan.HDBSCAN(
        min_cluster_size=size,
        min_samples=5,
```

```python
while True:
    clusterer = hdbscan.HDBSCAN(
        min_cluster_size=size,
        min_samples=5,
        cluster_selection_method='eom'
    )
    clusterer.fit(projection_t)
    labels = clusterer.labels_
    num_clusters = len(set(labels) - {-1})

    min_cluster_sizes_height.append(size)
    n_clusters_height.append(num_clusters)

    if num_clusters == 0:
        break
    size += 1

ax1.plot(min_cluster_sizes_height, n_clusters_height, marker='o', linestyle='-')
ax1.axvline(size, color='red', linestyle='--', label=f"Zero clusters @ min_cluster_size = {size}")
ax1.set_title('Minimum Cluster Size vs Number of Clusters (Height)')
ax1.set_xlabel('min_cluster_size')
ax1.set_ylabel('Number of Clusters')
ax1.legend()
ax1.grid(True, alpha=0.3)

# Second plot (Width)
# Initialize tracking lists
min_cluster_sizes_width = []
n_clusters_width = []
# Start sweep
size = 5
while True:
    clusterer = hdbscan.HDBSCAN(
        min_cluster_size=size,
        min_samples=5,
        cluster_selection_method='eom'
    )
```

```
    clusterer.fit(projection)
    labels = clusterer.labels_
    num_clusters = len(set(labels) - {-1})

    min_cluster_sizes_width.append(size)
    n_clusters_width.append(num_clusters)

    if num_clusters == 0:
        break
    size += 1

ax2.plot(min_cluster_sizes_width, n_clusters_width, marker='o', linestyle='-')
ax2.axvline(size, color='red', linestyle='--', label=f"Zero clusters @ min_cluster_size = {size}")
ax2.set_title('Minimum Cluster Size vs Number of Clusters (Width)')
ax2.set_xlabel('min_cluster_size')
ax2.set_ylabel('Number of Clusters')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```


Minimum Cluster Size vs Number of Clusters (Height)
Number of Clusters
min_cluster_size
Zero clusters @ min_cluster_size = 111
0
5
10
15
20
25
0
20
40
60
80
100


Minimum Cluster Size vs Number of Clusters (Width)
Number of Clusters
min_cluster_size
Zero clusters @ min_cluster_size = 179
0
2
4
6
8
10
12
14
16
0
25
50
75
100
125
150
175