Data Mining Assignment Report: Outlier Detection and Clustering Analysis

Authors:

Georgios Lazaridis AEM: 4419

Nicholas Christoforou AEM: 4479

Andreas Darsaklis AEM: 4403

Aristotle University of Thessaloniki

December 2025

1 Structure of Submission

This submission is presented as three complementary parts to make the division of work clear:

- **Part I — Traditional Implementation:** The dataset processing, cleaning scripts, the iterative K-Means implementation, and the visualizations live in the project folder and represent the core coursework deliverable.
- **Part II — Contributions to AI Clustering Tools:** Documentation and implementation of additional clustering tools (DBSCAN and Bayesian Gaussian Mixture) contributed to the open-source `DaMi` repository.
- **Part III — Reflections on AI Usage:** Commentary on how AI assistance was used throughout this project, its benefits, limitations, and lessons learned.

Part I: Traditional Implementation

Abstract

This report details the analysis of datasets with the objective of identifying outliers and performing data clustering. A combined approach involving robust data cleaning and an *Iterative K-Means* algorithm was implemented. This method focuses on refining centroid detection by filtering out noise before the final classification.

2 Introduction and Theoretical Background

Clustering is a fundamental technique in unsupervised learning used to group similar data points. The objective of this assignment was to process raw, corrupted data, identify anomalies (outliers), and partition the remaining data into distinct clusters.

2.1 The K-Means Algorithm

The primary algorithm selected for this task is **K-Means**. It aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean (centroid). The algorithm minimizes the Within-Cluster Sum of Squares (WCSS):

$$J = \sum_{i=1}^{k} \sum_{x \in S_i} ||x - \mu_i||^2 \tag{1}$$

Where μ_i is the mean of points in cluster S_i. Since K-Means relies on Euclidean distance, it is highly sensitive to the scale of the data axes and the presence of outliers, necessitating the custom methodology described below.

3 Data Pre-processing (Data Cleaning)

The raw data contained noise and corrupted records. A "defensive programming" approach was developed to ensure data integrity prior to analysis.

Specifically, the following steps were implemented:

- **Handling Corrupted Lines:** A mechanism was used to skip lines with an incorrect number of columns using the *on_bad_lines='skip'* parameter.
- **Coercion Errors:** Non-numeric values were coerced to `NaN` and subsequently removed.
- **Deduplication:** Duplicate records were removed to prevent density bias, which could negatively influence the positioning of K-Means centroids.

4 Analysis Methodology

4.1 Data Scaling

Due to the geometry of the data, a significant discrepancy was observed between the scales of the X and Y axes. Since K-Means relies on Euclidean distance, **Manual Scaling** was applied by dividing the Y axis by a calculated factor to equalize the range with the X axis.

Note on Standard Scalers: During the experimental phase, standard scaling libraries from Scikit-Learn, such as `MinMaxScaler` and `StandardScaler`, were tested. However, the clustering results were identical to those obtained via the manual scaling method. Consequently, the manual approach was retained for the final implementation to maintain better geometric interpretability of the specific datasets.

4.2 Iterative K-Means & Outlier Detection

To identify outliers and cluster the data effectively, an **Iterative K-Means** method was developed:

1. **Initial Run:** K-Means is executed on the scaled data to obtain an initial estimation.
2. **Refinement:** The algorithm identifies the "core" points of each cluster (the top 30% closest points to the initial centers).
3. **Re-Training:** K-Means is re-trained exclusively on these "clean" core points to calculate the **Refined Centroids**, ensuring they are not shifted by noise or outliers.
4. **Thresholding:** All points are classified based on these refined centroids. Points with a distance greater than a threshold of 3σ (Sigma) are classified as **Outliers**.

5 Results

The method was applied to the datasets `clean_data_a.csv` and `clean_data_b.csv`.

- **Dataset A:** The algorithm failed to correcly indetify all clusters due to scaling issues, resulting in misclassified outliers.
- **Dataset B:** The clusters were clearly separated, and the outliers (located at significant distances) were successfully detected and isolated from the main groups.

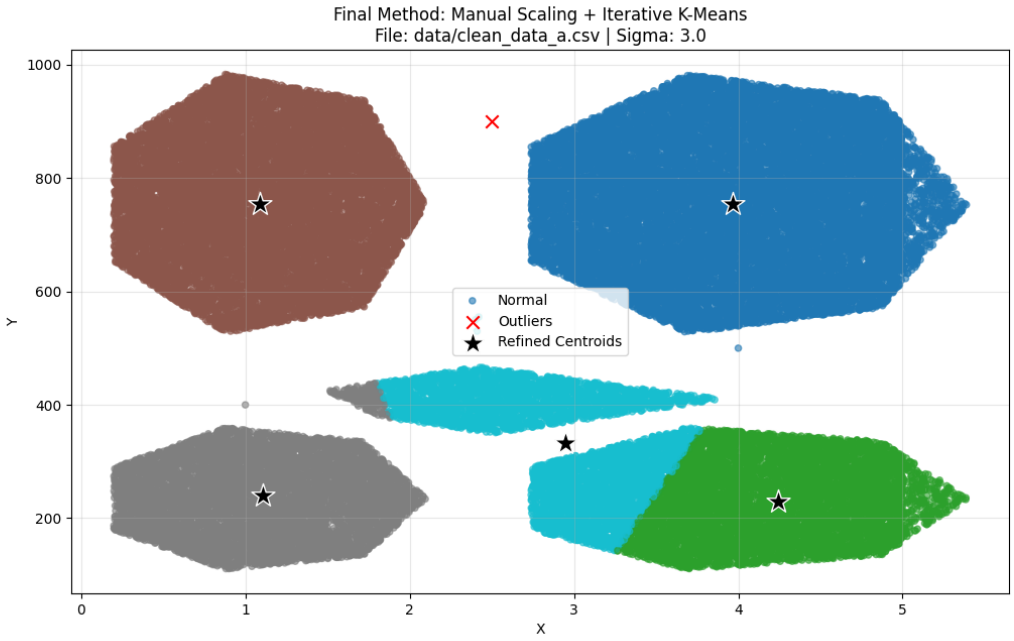

Figure 1: Clustering and Outliers on Dataset A

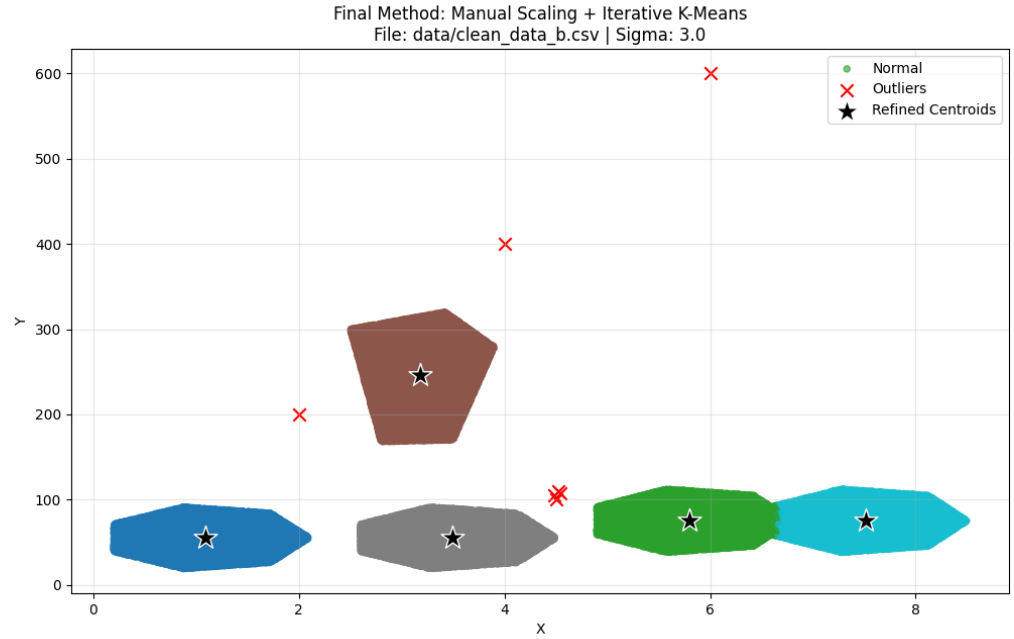

Figure 2: Clustering and Outliers on Dataset B

6 Implementation Details

The project was implemented using the **Python** programming language.

6.1 Environment and Dependencies

To ensure that the results can be replicated on any machine, a Virtual Environment (`venv`) was utilized to isolate the project dependencies. A `requirements.txt` file is included in the repository, allowing for the automatic installation of the exact library versions used during development.

6.2 Libraries Used

The following libraries were utilized for the implementation:

- `pandas`: For data loading, manipulation, and cleaning.
- `numpy`: For numerical operations and array handling.
- `matplotlib`: For data visualization and plotting results.
- `scikit-learn (sklearn)`: For the K-Means clustering algorithm.

6.3 Source Code Repository

The complete source code, including the cleaning scripts, the clustering algorithm, and the final visualization outputs, is hosted on GitHub at the following link:

https://github.com/lazoulios/data-mining-and-clustering

7 Conclusion

The application of the Iterative K-Means method, combined with appropriate data scaling, proved highly effective. The process successfully separated noise from useful data and achieved good centroid detection, confirming the robustness of the methodology for these specific datasets, while highlighting areas for improvement in extreme scaling issues with cluster shapes.

Part II: Contributions to AI Clustering Tools

As part of the bonus work for this assignment, we contributed documentation and implementation for two additional clustering algorithms to the open-source `DaMi` repository available at:
https://github.com/Bilpapster/DaMi

8 DBSCAN Documentation

We created comprehensive documentation for the DBSCAN (Density-Based Spatial Clustering of Applications with Noise) tool. DBSCAN is a density-based clustering algorithm that groups together points that are closely packed together, marking points that lie alone in low-density regions as outliers. Unlike K-Means, which requires the number of clusters to be specified in advance, DBSCAN automatically discovers clusters based on local density.

The algorithm operates using two key parameters:

- `eps`: Maximum distance between two samples for one to be considered in the neighborhood of the other.
- `min_samples`: Minimum number of samples in a neighborhood for a point to be considered a core point.

The interplay between these parameters determines the sensitivity of the algorithm to cluster density and noise.

DBSCAN offers several significant advantages over traditional centroid-based methods:

- Does not require pre-specifying the number of clusters.
- Can find arbitrarily shaped clusters (not limited to spherical shapes).
- Naturally robust to outliers and noise.
- Can handle clusters of varying densities.

However, the method has notable limitations:

- Highly sensitive to parameter selection (`eps` and `min_samples`).
- Struggles with high-dimensional data due to the curse of dimensionality.
- Computationally intensive: O(n log n) to $O(n^2)$ complexity.
- Different parameter settings can produce significantly different results.

DBSCAN is most appropriate when the number of clusters is unknown, the data contains noise and outliers that should be explicitly identified, clusters have irregular shapes that would confound centroid-based methods, or when dealing with datasets that exhibit varying cluster densities. Conversely, it should be avoided when working with high-dimensional data, when clusters are of similar size and density (where K-Means may be more efficient), or when computational efficiency is a primary concern.

9 Bayesian Gaussian Mixture Tool

We implemented and documented a new clustering tool using Scikit-Learn's `BayesianGaussianMixture`, which provides a probabilistic mixture model with automatic complexity selection via a Dirichlet process prior. This approach represents a significant departure from both K-Means and DBSCAN by treating clustering as a probabilistic inference problem rather than a deterministic assignment task.

The tool offers several key capabilities that distinguish it from traditional methods. Most notably, it can automatically infer the effective number of components, eliminating the need for manual selection of cluster count. It provides probabilistic (soft) cluster assignments with confidence metrics, allowing users to quantify uncertainty in the clustering results. The implementation is robust to over-specifying the initial number of components, as the Bayesian framework naturally prunes unused components through the prior. The tool supports configurable covariance types (full, tied, diagonal, or spherical) to match different cluster shapes, and exports comprehensive results to CSV including cluster labels, assignment probabilities, and metadata for reproducibility.

The tool accepts several configuration parameters:

- `file_path`: Path to the CSV file containing the dataset.
- `n_components`: Maximum number of mixture components (default: 3).
- `covariance_type`: Covariance structure ('full', 'tied', 'diag', or 'spherical').
- `weight_concentration_prior_type`: Prior type ('dirichlet_process' or 'dirichlet_distribution').
- `random_state`: Random seed for reproducibility.

The Bayesian approach provides several advantages:

- Automatically infers the effective number of components through the prior.
- Provides probabilistic assignments for uncertainty quantification.
- Handles overlapping clusters naturally through the mixture model.
- Robust when the initial component count is over-specified.
- Can serve as a generative density model for sampling or anomaly scoring.

These properties make it particularly valuable when cluster structure is ambiguous or when quantifying uncertainty is important.

However, the method has computational and practical limitations:

- Significantly more expensive than K-Means (time and memory).
- Requires careful tuning of priors and parameters.
- Not suitable for very large datasets due to memory/runtime constraints.
- Requires prior dimensionality reduction for high-dimensional data.
- Strong density variations or noise may mislead the model.

The tool is most appropriate when the true number of clusters is unknown and automatic inference is desired, soft probabilistic cluster assignments are required for downstream analysis, datasets contain overlapping clusters or clusters of varying sizes and densities, or when initial over-specification of components is acceptable with automatic pruning. It should be avoided when working with very large datasets where runtime and memory are constrained, high-dimensional data without prior dimensionality reduction, extremely fast deterministic hard-centroid clustering is necessary, or when data contains strong density variations or noise that violate the Gaussian mixture assumptions.

The implementation automatically standardizes input data using `StandardScaler`, fits the Bayesian Gaussian Mixture model, and exports results with both hard labels via maximum a posteriori (MAP) assignment and soft probabilities for each point, providing a complete picture of the clustering solution.

9.1 Testing and Integration Process

The Bayesian Gaussian Mixture tool was initially developed and tested on an external, independent clustering problem to validate its functionality and ensure correct implementation. This external testing phase allowed us to verify that the algorithm worked correctly, that parameter handling was robust, and that output formatting met our requirements before attempting integration with the DaMi repository.

Once the tool demonstrated reliable performance on the external test case, we proceeded to integrate it into the DaMi repository. During integration, we encountered challenges with the overall DaMi execution environment, which required careful parameterization and configuration adjustments. The tool was adapted to fit the DaMi framework by ensuring proper parameter passing, correct file path handling, and compatibility with the repository's existing infrastructure.

Still, we were **unable to confirm if the algorithm correctly executes** within the DaMi environment due to the inability to run the full DaMi toolchain locally.

10 Libraries and Dependencies

The DBSCAN and Bayesian Gaussian Mixture tools rely on established scientific Python libraries to provide robust clustering functionality. Both tools utilize the following core dependencies:

- `scikit-learn (sklearn)`: Provides implementations of DBSCAN via `sklearn.cluster.DBSCAN` and Bayesian Gaussian Mixture via `sklearn.mixture.BayesianGaussianMixture`. These implementations are highly optimized and widely tested in production environments.
- `pandas`: Used for reading CSV files, data manipulation, and exporting results to CSV format for downstream analysis and reporting.
- `numpy`: Provides the numerical array operations and mathematical functions required for data standardization, distance calculations, and probability computations.
- `StandardScaler (from sklearn.preprocessing)`: Employed to standardize input features, ensuring that variables on different scales do not bias the clustering algorithms.

These dependencies are minimal and industry-standard, making the tools easy to install and reproduce on any system with a standard Python scientific computing environment. The tools are designed to integrate seamlessly with existing data pipelines and can be easily extended with additional preprocessing or post-processing steps as needed.

Part III: Reflections on the Use of AI in This Project

To evaluate the capabilities and limitations of Large Language Models (LLMs) in data mining tasks, we conducted three experiments using an LLM assistant. These experiments aimed to assess whether LLMs can provide meaningful contributions to data analysis beyond documentation and code generation.

11 Experiment 1: LLM Evaluation of Our Implementation

We provided the LLM with our complete source code (`data_pre_processing.py` and `clustering.py`) and requested a critical evaluation as if reviewing a student assignment. The LLM's assessment proved surprisingly thorough and technically sound.

The LLM correctly identified the strengths of our implementation:

- **Robust error handling:** The `on_bad_lines='skip'` approach for corrupted data.
- **Clear cleaning pipeline:** Coercion to numeric types, NaN removal, and deduplication.
- **Well-motivated methodology:** The iterative K-Means approach for robustness against outliers.
- **Effective visualization:** Clear separation of clusters, outliers, and centroids.

More importantly, the LLM identified several valid concerns and limitations in our methodology:

- **Scaling vulnerability:** The `scaling_factor = max_y / max_x` approach is sensitive to outliers and fails if `max_x = 0`. Suggested alternatives: median-based scaling, IQR ratios, or `RobustScaler`.
- **Arbitrary thresholds:** The 30% closest points selection is heuristic and may disadvantage sparse or irregular clusters.
- **Outlier detection fragility:** Using `mean` + $\sigma \times$ `std` assumes normal distribution. Outliers inflate both mean and std, causing the threshold to rise and outliers to hide themselves—a classic bootstrapping problem.
- **Missing validation:** No justification for `k=5` clusters (elbow plot or silhouette analysis recommended).

Overall, the LLM demonstrated strong capability in code review and methodological critique. Its suggestions were technically sound and aligned with best practices in data mining literature. This experiment indicates that LLMs can serve as valuable "second reviewers" for catching methodological weaknesses and suggesting improvements, particularly in identifying implicit assumptions and edge cases that might be overlooked during initial implementation.

12 Experiment 2: Can an LLM Detect Outliers Without Algorithms?

To test whether an LLM can perform actual data analysis, we challenged it to identify outliers in our cleaned datasets (`clean_data_a.csv` and `clean_data_b.csv`) using only statistical observation—no clustering algorithms, no distance calculations, and no K-Means. This experiment was designed to probe the fundamental capabilities and limitations of LLMs in quantitative analysis.

The LLM approached the task by examining the range of x and y values to understand the data's extent, attempting visual inspection of value distributions through the raw numbers, and trying to identify "suspicious" extreme values based on their position relative to the observed ranges.

For Dataset A, the LLM correctly detected that x ranges from approximately 0.2 to 5.4 and y ranges from approximately 111 to 983. It identified potential outliers as points with very small x (≈ 0.2) or very large x (> 5), especially when combined with extreme y values near 983 or 111. However, the LLM encountered a critical limitation: it could not distinguish between points belonging to sparse clusters and true outliers. Without geometric distance calculations or density estimation, the LLM had no principled way to determine whether an extreme point was simply part of a low-density cluster or genuinely anomalous.

For Dataset B, the LLM detected x ranging from 0.2 to 8.5 and y from 19 to 600. It flagged points with $y \approx 600$ or $y \approx 19$ as suspicious, and noted points with $x > 8$ as potential outliers. Yet the same fundamental limitation persisted—the ambiguity between legitimate sparse regions and true anomalies could not be resolved without algorithmic analysis.

Table 1: Summary of LLM Outlier Detection Findings

Dataset	**X Range**	**Y Range**	**Suspected Outliers**
Dataset A	0.2 to 5.4	111 to 983	Very small x (≈ 0.2) or very large x (> 5) combined with extreme y values (near 983 or 111)
Dataset B	0.2 to 8.5	19 to 600	Points with $y \approx 600$ or $y \approx 19$; points with $x > 8$
Critical Limitation (Both Datasets): Cannot distinguish			

The LLM provided its own candid self-assessment (translated from Greek): *"Although the LLM can indicate potentially extreme regions based on general observations, it is unable to accurately determine which points constitute true outliers. The lack of geometric and algorithmic analysis leads to ambiguity, particularly in data with multiple clusters or uneven density. Consequently, LLMs are not suitable for reliable outlier detection but can only be used supplementarily for qualitative interpretation."*

This experiment clearly demonstrates a fundamental limitation of LLMs in data mining tasks. Without access to computational geometry, distance metrics, or density estimation algorithms, LLMs cannot replace proper algorithmic methods. They can provide high-level intuitions about what *might* be unusual in a dataset, but they fail at precise, quantitative analysis that requires numerical computation. This reinforces the necessity of proper algorithmic implementations like our iterative K-Means with distance-based thresholding, which can compute exact distances, estimate local densities, and apply statistically motivated thresholds to identify outliers reliably.

13 Experiment 3: LLM-Generated Datasets and Algorithm Performance

We requested the LLM to generate three datasets to further test our algorithm we developed in Part I. The prompt specified that each dataset should contain 35-45 data points with two dimensions (x and y) and be designed to test different failure modes and edge cases of clustering algorithms.

The original datasets were provided as references to guide the LLM in creating similar but distinct datasets. The LLM successfully generated three CSV datasets (`llm_data_1.csv`, `llm_data_2.csv`, and `llm_data_3.csv`), each containing 35-45 data points with two dimensions (x and y).

We then executed our iterative K-Means implementation (from Part I) on all three LLM-generated datasets using `k=5` clusters and our standard scaling parameters. The algorithm clustered each dataset and identified outliers based on the 3σ threshold. The results are visualized in the following figures:

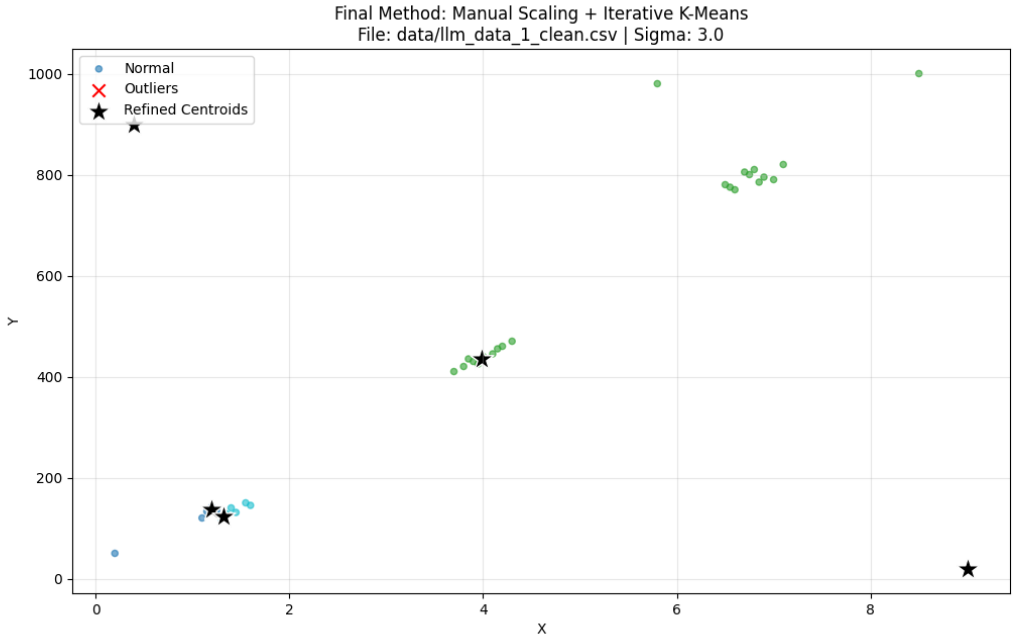

Figure 3: Clustering Results on LLM-Generated Dataset 1

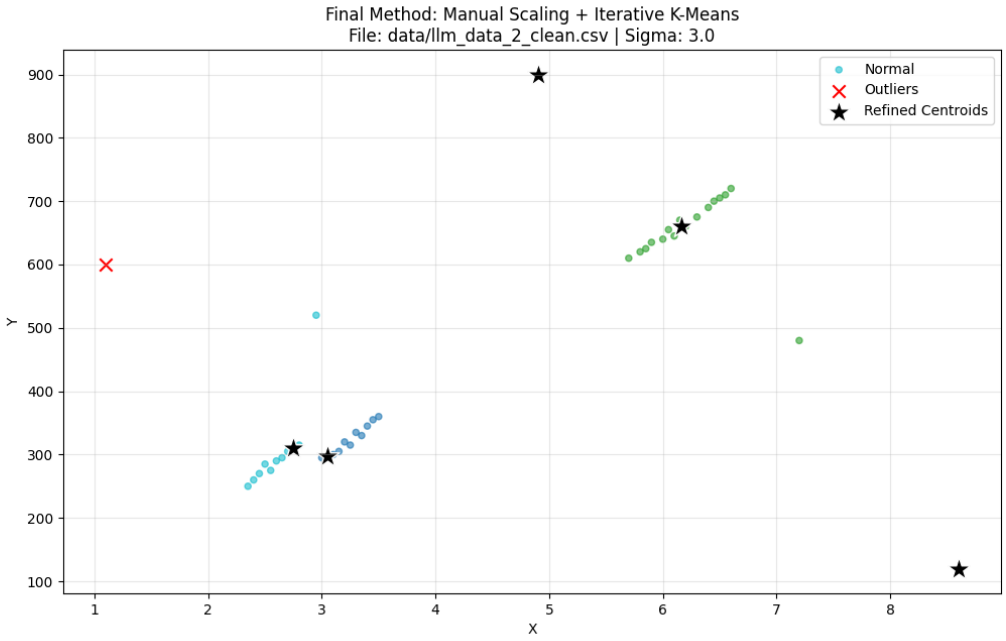

Figure 4: Clustering Results on LLM-Generated Dataset 2

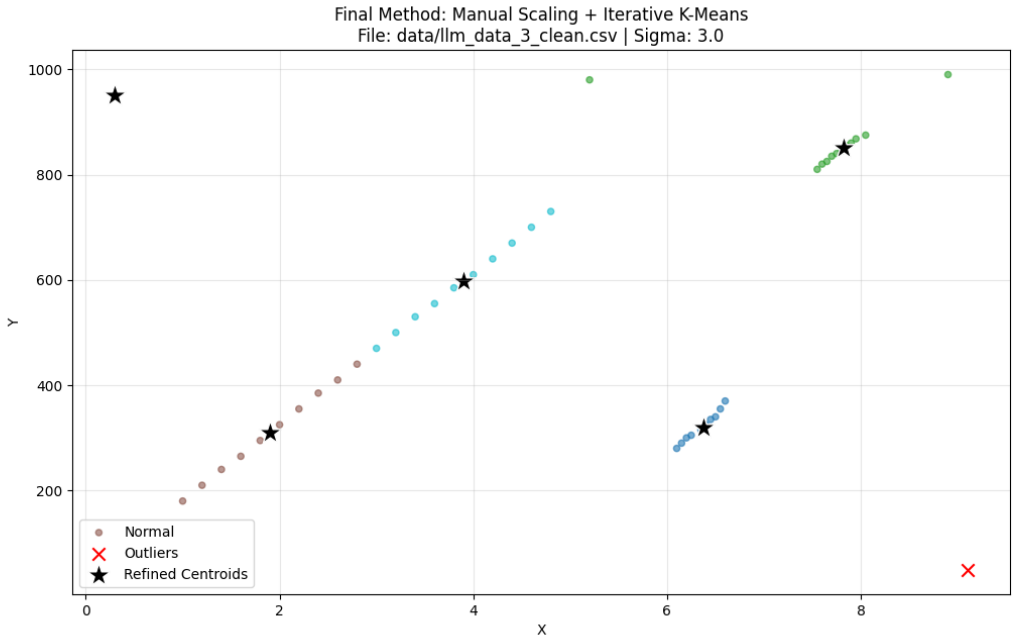

Figure 5: Clustering Results on LLM-Generated Dataset 3

Key Observations:

- **Dataset 1:** The algorithm failed to identify the three main clusters and flagged obvious outliers. Performance was bad on this straightforward case.

- **Dataset 2:** Same results were observed.
- **Dataset 3:** Same results were observed.

  Assessment: This experiment demonstrates that LLMs can generate datasets that expose both strengths and weaknesses in clustering algorithms. Still the datasets generated did not seem to fit the k=5 limitation of our algorithm, leading to poor performance.

14 General Reflections on AI Usage

Based on our experiments and overall experience with LLM assistance throughout this project, we can draw several important conclusions about the appropriate role of AI in data mining work.

LLMs demonstrate clear strengths in certain areas:

- **Code review:** Identification of bugs and methodological weaknesses.
- **Documentation:** Clear explanations of complex concepts and comparative descriptions.
- **Literature suggestions:** Alternative approaches and relevant references.
- **Writing acceleration:** Rapid synthesis and report generation.

However, LLMs have fundamental limitations that must be respected:

- **No numerical computation:** Cannot calculate distances, densities, or perform data analysis.
- **No quantitative decisions:** Cannot distinguish outliers from sparse cluster members.
- **No geometric understanding:** Rely on heuristic pattern matching, not true geometric properties.
- **Cannot validate methods:** Require code execution to verify if approaches work.

These observations lead to several best practices for AI-assisted data mining:

- Use LLMs as advisors and documentation assistants, not as analysts.
- Validate all LLM suggestions with empirical tests before acceptance.
- Maintain human oversight for all technical decisions.
- Document AI contributions transparently for reproducibility.
- Treat LLM outputs as hypotheses requiring verification.