Incremental K-Means Clustering Using Variance Change Proportion: A Dynamic Approach for Real-Time Data Integration

Peter Krompiewski, Asit Patel, Muhammad Ibrahim, Eddie Butkaliuk

COMP 4710, University of Manitoba

Winnipeg, Canada

krompiep@myumanitoba.ca

patela8@myumanitoba.ca

ibrahi30@myumanitoba.ca

butkalie@myumanitoba.ca

***Abstract*— In today's data-driven world, extracting valuable insights from large amounts of data has become increasingly important. As data grows in both volume and complexity, many businesses require efficient methods to understand and cluster (categorize) it for analytical reasons. As such, this paper introduces an incremental k-means algorithm that addresses certain shortcomings of traditional k-means clustering and its extensions by introducing new data points in real time. Our incremental clustering approach dynamically adapts cluster centroids without requiring a complete re-clustering. Additionally, the proposed method uses adaptive thresholds to make decisions efficiently, keeping a reasonable balance between computational costs and clustering accuracy. Many of our contributions have been inspired by those existing k-means variants, as we try to improve upon what works, rather than inventing a whole new method. Detailed evaluations highlight its effectiveness in handling large, dynamic datasets, significantly reducing time complexity while maintaining high quality clustering results. Real-world applications like customer segmentation, document classification, and image analysis highlight how this approach can provide real-time insights and transform various industries.**

***Keywords* — K-Means, Clustering, Dynamic, Incremental Clustering, Real-Time Data, Centroids. Data Mining, Algorithms.**

I. Introduction

Clustering is a popular data mining technique which groups similar data points into meaningful clusters (groups), with the **k-means algorithm** standing out for its efficiency in static datasets. It is also widely accessible and relatively easy to implement. However, it has limitations which become apparent in dynamic environments, where continuously arriving data requires frequent re-clustering, ultimately slowing the algorithm down drastically.

A. Existing K-Means Algorithms

The k-means algorithm works by initializing cluster **centroids** (centers), typically chosen randomly, and then iteratively assigning data points to the closest centroid [13]. After assignment, the centroids are recalculated as the mean of the points within each cluster [13]. This process is repeated until the centroids no longer change significantly between iterations [13]. This method works extremely well, however, its limitations arise when applied to **real-time environments**, where new data points arrive continually. In this situation, traditional k-means requires re-clustering the entire dataset each time new data comes in, which can be computationally expensive and time consuming.

B. Motivations

This full re-clustering process becomes increasingly inefficient as the dataset grows, resulting in slower processing times [1]. For example, in a dataset of 100 existing data points organized into 5 clusters, the addition of 100 new data points would require recalculating all clusters from scratch. This issue becomes exponentially worse at larger scales, where datasets can contain thousands or even millions of data points. Ultimately, this inefficiency often leads users to abandon k-means entirely in favor of alternative solutions better suited to real-time addition of data. However, rather than discarding k-means, we propose building on its foundation to enhance its ability to handle dynamic datasets more effectively.

Incremental clustering offers a feasible solution by enabling real-time integration of new data points, either by adjusting existing clusters or creating new ones. The aim is to develop an algorithm that achieves a balance between clustering accuracy and computational efficiency, resulting in a "good enough" adjustment suitable for real-time applications. This approach has the potential to enhance the utility of clustering techniques in such situations.

C. Potential Real-Life Applications

Incremental clustering has a wide range of applications in fields where real-time insights are crucial for decision-making. For example, in **customer segmentation**, retail and e-commerce platforms can continuously update customer profiles as new transaction data arrives, allowing businesses to create personalized recommendations based on the most current customer behavior [5]**.** This dynamic adjustment helps businesses stay responsive to changes in customer preferences, improving customer engagement and satisfaction.

Additionally, incremental clustering is especially valuable for **document classification** and clustering, as it allows the efficient addition of new documents into existing categories without requiring a full re-clustering process. This capability is crucial in dynamic systems like news portals, academic databases, and corporate knowledge management platforms, where new content is continuously generated [11]. This approach ensures that document classification remains up-to-date without the need for reprocessing the entire dataset, improving both speed and accuracy.

Furthermore, this technique is also proving to be invaluable for applications such as **image segmentation**, where datasets evolve continuously. For instance, in domains such as autonomous vehicles, medical imaging, and satellite analysis, large volumes of data are generated in real time [12]. Once again, incremental clustering would allow algorithms to adapt in real-time by updating existing clusters or forming new ones. This adaptability not only enhances efficiency but also ensures the relevance and accuracy of segmentation as more data is introduced. As discussed in the context of document clustering, where changing datasets need flexible methods to keep related text grouped together, similar ideas in image segmentation can make processing faster while still being efficient. By allowing the system to adapt in real time to new image data, these methods enhance the ability to detect and analyze objects or features in dynamic environments.

Finally, recent advances in technology, such as cloud computing and the **Internet of Things** (IoT), have increased the need for efficient incremental clustering methods. IoT-enabled systems, for example, continuously generate large streams of data from interconnected devices in industries like manufacturing and autonomous systems [1]. In this context, the ability to adapt clustering models dynamically without reprocessing the entire dataset is essential. Incremental clustering methods aim to address this challenge by updating only the affected clusters when new data points are introduced, significantly reducing computational costs.

D. Contributions of this Work

This work introduces an extension of the k-means clustering algorithm, using it as a framework to overcome its challenges in managing dynamic datasets. Unlike traditional methods that require complete re-clustering when new data points are introduced, the proposed algorithm is designed to integrate new data incrementally, allowing real-time clustering adjustments. This new approach focuses on dynamically adapting cluster centers and efficiently determining whether to assign incoming data to existing clusters or create new ones. A major aspect of the proposed method is the use of **adaptive thresholds**, which help decide whether new clusters are necessary or when existing clusters should be adjusted. These thresholds help balance the use of computing resources with the accuracy of the clustering.

To validate the proposed algorithm, a detailed performance evaluation is conducted. This includes comparisons with the traditional k-means algorithm across several metrics, such as clustering quality, time complexity, and scalability. The results highlight the practical advantages of the new method, especially in scenarios involving continuously growing datasets. Ultimately, our work contributes to the field by developing a time-efficient clustering algorithm that greatly minimizes computational costs compared to traditional re-clustering techniques.

II. Background & Related Works

This section explores the foundational concepts and existing advancements in k-means clustering, focusing on their advantages and drawbacks. Additionally, it introduces

the unique contributions of our proposed approach in overcoming these drawbacks.

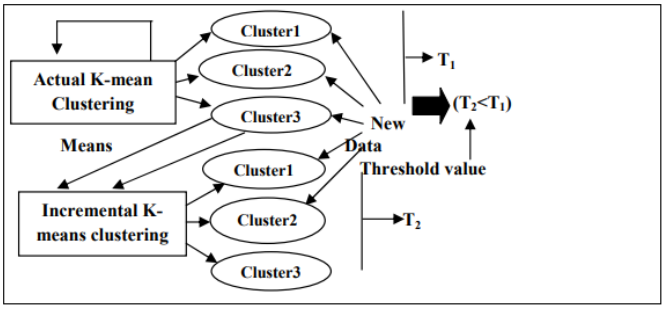


Fig 1 - Visualizing k-means vs Incremental k-means [2]

A. K-Means Clustering and Its Extensions

K-means clustering is a widely used centroid-based algorithm designed to partition data into a predefined number of clusters referred to as ***K*** [2]. By assigning data points to the nearest cluster and recalculating cluster centroids, the algorithm continues until convergence is achieved. Convergence in K-means clustering happens when the algorithm stops making changes. This means the data points stay in the same clusters, and the center of each cluster no longer moves. At this point, the algorithm has finished and found the best grouping of the data. The k-means algorithm finds locally optimal solutions with respect to the clustering error. It is an iterative algorithm that begins with K cluster centers [13]. In its simplest form, these K centers are usually chosen randomly from the set of data points x [13]. The algorithm then repeats in two steps, assigning data points to clusters and optimizing the cluster centers until it converges. This includes the data assignment and the center optimization steps [13]. At the assignment step, every data point x_i is assigned to the cluster C_j with the nearest center m_j.

$$j = \arg\min_k ||x_i - m_k||^2$$

Fig 2 - k-means assignment step [13]

At the optimization step, each center is updated to the mean of all data points assigned to its cluster.

$$m_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$$

Fig 3 - k-means optimization step [13]

As previously stated, k-means has limitations when faced with dynamic datasets. Several extensions to traditional k-means have emerged to address these constraints. For instance, **Mini-batch k-means** processes small, randomly selected subsets of data, thus reducing memory usage [6]. For any given dataset $S = \{x_i\}$, $x_i \in \mathbb{R}^{m \times n}$ represents a data record that is an n-dimensional real vector, and m indicates the number of records contained in the dataset [6]. The clustering problem is to find the centers $C = \{c_1, L, c_K\}$, $c_i \in \mathbb{R}^{m \times n}$ such that the following function is minimized:

$$\min \sum_{x_i \in S} \| f(C, x_i) - x_i \|^2$$

Fig 4 - Mini-batch k-means [6]

where, $f(C, xi)$ returns the closest cluster center $c_i \in C$ to record x_j, and $|C| = K$ is the number of clusters we want to find [6].

Fuzzy k-means is an alternative approach that gives data points partial memberships in multiple clusters, making it useful for datasets with overlapping features [5]. In **fuzzy clustering**, each point has a level of membership in each cluster, instead of being fully assigned to just one. So, points at the edge of a cluster belong less strongly to it compared to points in the center [5]. For each point x we have a coefficient giving the degree of being in the k^{th} cluster $u_k(x)$. Usually, the sum of those coefficients is defined to be:

$$\forall x \sum_{k=1}^{\text{num.clusters}} u_k(x) = 1.$$

Fig 5 - Fuzzy k-means coefficient [5]

With fuzzy k-means, the centroid of a cluster is the mean of all points, weighted by their degree of belonging to the cluster:

$$\text{center}_k = \frac{\sum_x u_k(x)^m x}{\sum_x u_k(x)^m}.$$

Fig 6 - Fuzzy k-means centroid calculation [5]

The degree of belonging is related to the inverse of the distance to the cluster center:

$$u_k(x) = \frac{1}{d(\text{center}_k, x)},$$

Fig 7 - Fuzzy k-means degree of belonging [5]

then the coefficients are normalized and adjusted to reflect partial membership in multiple clusters with a real parameter $m > 1$, so that their sum is 1:

$$u_k(x) = \frac{1}{\sum_j \left(\frac{d(\text{center}_k, x)}{d(\text{center}_j, x)} \right)^{2/(m-1)}}.$$

Fig 8 - Fuzzy k-means coefficients normalized [5]

For $m = 2$, it is equivalent to normalising the coefficient linearly to make their sum 1. When m is close to 1, the cluster center closest to the point is given much more weight than the others, and the algorithm is similar to k-means [5].

Streaming k-means is an altered version of the k-means algorithm which handles continuous data streams. When new data arrives, it updates cluster centroids dynamically by assigning incoming points to the closest cluster and recalculates centroids based on weighted averages [4]. This process allows the algorithm to keep up with changing data without re-clustering the entire dataset. Although very simple, the approach is efficient and scalable, using techniques like mini-batch updates or outlier discarding for better performance. However, streaming k-means often sacrifices accuracy, especially when data patterns change quickly, as the cluster updates might lag behind the true distribution [4].

Dynamic block-based k-means is yet another extension of traditional k-means. Unlike streaming k-means, it can adjust the number of clusters as the data changes. It uses methods to combine clusters that are close together, split clusters with wide spread, or simply remove clusters with too few points, following principles of the ISODATA algorithm [3]. The main difference is that data is grouped into blocks, processed, and then clustered, for better memory handling. Additionally, dynamic k-means uses certain thresholds like distortion, to decide when to add or remove clusters. A two-phase process is often used, where centroids from earlier blocks are used to essentially initialize the next ones, aiming to keep the clustering consistent as data changes [3]. Rules for quality and the number of clusters make sure dynamic k-means keeps enough clusters for accuracy while still being relatively efficient [3].

B. Benefits of Existing Works

Advancements in k-means clustering have addressed some of its limitations in its base form. Mini-batch k-means is one extension of the traditional k-means algorithm, aimed to efficiently handle large datasets by processing small, randomly selected subsets (mini-batches) of data. The key principle of using mini-batches is to avoid handling the entire dataset at once, which could either be extremely slow, or simply not work at all. This approach uses less memory and processing power, making it a viable option to handle larger datasets while maintaining similar clustering quality [6]. It also uses smart optimization techniques to work faster, which makes mini-batch k-means a solid fit for real-time tasks such as web analytics and recommendation systems.

Similarly, dynamic block-based k-means also provides a significant advantage in handling large or memory-limited datasets by processing data in smaller, manageable blocks. However, the key difference here is that data is processed incrementally in fixed blocks and cluster centroids are updated as each block is processed. Contrary to the regular k-means, which requires the entire dataset to be available and accessible in memory, dynamic block k-means enables clustering on subsets of data, making it scalable and efficient for datasets that exceed memory limitations [3]. It updates cluster centroids as each block is processed and adapts by splitting or merging clusters based on how spread out the data is or certain thresholds [3]. This makes it ideal for situations where the data is too big to handle all at once but still requires flexible and adaptive clustering.

Lastly, Fuzzy k-means is extremely beneficial in situations where there is uncertainty linked to data. Often, it's difficult to categorize a data point, as it can make sense to place it within multiple clusters. For example, in the customer segmentation department, a customer can belong to various groups like "budget-friendly" and "decor-oriented", which can lead to ambiguity during clustering [5]. With fuzzy k-means, it becomes possible to assign a degree of belonging to both clusters. Therefore, during analysis of such data, we can retain complete information about that customer, including all the groups that it is a member of.

C. Shortcomings of Existing Works

Despite their strengths, these existing techniques have many shortcomings. Most algorithms still rely on complete re-clustering upon receiving new data, which by now, we know is inefficient. Techniques like mini-batch and streaming k-means were seen to improve speed and efficiency by handling data in smaller chunks, but this comes at the cost of neglecting certain outliers completely. This diminishes the potential for these algorithms to be used in tasks where precision is crucial. For example, systems that are highly sensitive to outliers can't rely on streaming k-means as it tends to ignore these small but important details.

In mini-batch k-means, for instance, the use of randomly selected subsets of data can lead to suboptimal cluster assignments, as the algorithm may fail to fully capture the true structure of the data [6]. For example, if a mini-batch contains data points that are not representative of the overall distribution, the algorithm might assign them to clusters that don't reflect the true cluster structure. Additionally, mini-batches are selected randomly, so the results can vary depending on which data points are sampled [6].

Likewise, streaming k-means processes continuous data in real-time, but this often results in less accurate clusters due to the limited amount of data considered at any given time, especially when noise or outliers are present. This can be a big issue in areas like medical diagnostics, where miniscule inaccuracies in clustering could have serious consequences [6]. In critical fields like this, reliable and accurate clusters outweigh the desire for speed, therefore it is crucial to find a balance between the two [6].

As seen, dynamic k-means offers flexibility by adjusting the number of clusters as data changes, making it useful for real-time applications. However, its implementations vary widely, with no standardized approach, leading to inconsistencies in computational efficiency and clustering quality [3]. Some variants prioritize speed, sacrificing accuracy, while others focus on precision at the cost of higher computational demands. This lack of a consistent, balanced implementation makes dynamic k-means challenging to apply in scenarios where both efficiency and quality are crucial [3]. Moreover, the implementation of blocks can slow down real-time processing because new data has to be grouped into blocks before it can be analyzed [3]. This delay can be a problem for systems like fraud detection, which require urgent action to catch suspicious activities. Furthermore, deciding when to create, merge, or adjust clusters can be difficult or sometimes impossible in real-world scenarios, often causing inconsistent results.

D. How Our Work Addresses These Issues

The proposed incremental k-means algorithm builds upon the strengths of existing methods while directly addressing their limitations. By incorporating an efficient technique for integrating new data points, the algorithm delays the need for full re-clustering for as long as possible, thereby reducing computational overhead. The previously discussed algorithms each tackle specific aspects of the challenges we aimed to address, but none eliminate all those issues. Our algorithm is designed to meet all these requirements while maintaining the same worst-case complexity as traditional k-means, ultimately offering significantly improved average-case performance.

One of the key features absent in previous algorithms include dynamic cluster updates, which allow centroids to be continuously adjusted as new data is added, ensuring that clusters remain accurate and relevant over time. Additionally, our approach introduces automatic real-time reclustering based on a threshold metric, designed to balance accuracy and computational efficiency. This threshold metric, called the **Variance Change Proportion**, is our unique contribution. It establishes an allowable level of variance deviation within clusters, making it computationally efficient. This metric plays a pivotal role in our incremental algorithm, as dynamic point additions can only be sustained up to a certain point before clusters diverge too far from their optimal variance. When this threshold is exceeded, the algorithm triggers reclustering, ensuring accuracy while delaying the process as long as feasible to maximize efficiency. Dynamic threshold management ensures that cluster adjustments are both precise and resource-efficient, providing a balanced approach to clustering quality and computational requirements.

Additionally, the proposed work includes a comprehensive evaluation of time complexity, clustering accuracy, and scalability, offering practical insights and a strong framework for real-world applications. This combination of features positions our approach as a significant improvement over current methodologies in dynamic clustering scenarios.

III. Proposed Method

A. Key features and Innovations

Before exploring the proposed method, it's important to briefly highlight the specific point in the existing algorithm where it tends to slow down. This will help us better understand how the proposed method addresses this issue.

The K-means algorithm itself does many iterations of two key steps: *Assignment* and *Optimization.* These two steps are performed on the data points until *Convergence* is reached. During *Assignment*, each data point is assigned to the nearest centroid. Afterwards, in the *Optimization* step, the centroids are recalculated based on the new groupings. This cycle continues repeatedly until Convergence is reached. This process is already computationally heavy on its own, and yet the complexity increases significantly when additional heuristics are introduced. For example, one common practice is to run the K-means algorithm multiple times using different initial centroid positions [7]. This is done to minimize the overall cluster variance, aiming to result in a better solution, as the final result can vary depending on the starting points. Another layer of complexity comes from methods like the Elbow Method, which involves running the entire K-means algorithm repeatedly for various numbers of clusters, typically starting from $k = 2$, $k = 3$, $k = 4$, and so on. This process continues until the optimal number of clusters (k) is identified based on the method's criteria, such as identifying where the "elbow" or point of diminishing returns occurs in the graph of explained variance [8]. As one could observe, each additional heuristic evidently compounds the computational load, increasing the algorithm's complexity drastically. This is precisely why the algorithm can quickly become impractical for large datasets, as the time and resources required grow at rapid rates. Thus, while these methods improve accuracy and reliability of the results, they come at the cost of significant computation costs.

To further understand the computational challenges of the K-means algorithm, it's crucial to consider its actual time complexity, as this provides insight into why scalability becomes an issue. Conventionally, K-means is classified as an NP-hard problem, meaning it's computationally unmanageable to solve optimally in reasonable time for large datasets. Specifically, the time complexity of the algorithm is $O(NTK)$, where N is the number of data points, T is the number of iterations required for convergence, and K is the number of clusters. Therefore, this implies that as the number of data points, clusters or iterations increase, the computational time grows proportionally [9].

Additionally, the space complexity of K-means is $O(N(D + K))$, where D is the number of dimensions in the data. Some modern datasets often involve several dimensions, such as in machine learning applications, where data points may include preferences or behaviour [10]. However, for simplicity, we conducted our research in just 2 dimensions, the X and Y axis.

As discussed earlier, the traditional algorithm involves a computationally expensive process for clustering a static dataset. However, in dynamic datasets where data is constantly flowing, reclustering from scratch every time a new data point comes in is not only inefficient but also computationally heavy. This challenge is what our method aims to solve. Instead of reprocessing the entire dataset with every change, our approach introduces data points incrementally as they arrive in the data stream. Concurrently, we continuously verify the *correctness* of the clusters through the change of variance within each cluster. When the variance surpasses some predefined threshold, the variance structure is now considered *incorrect*, where reclustering gets triggered.

The fundamentals of this approach is a metric we call the **Variance Change Proportion (VCP)**. This metric tracks the variance of each cluster after every new data point is added and compares it to the cluster's initial variance. If the difference between the current and initial variances exceeds a certain percentage or proportion, reclustering then happens. This is done to assure clusters remain accurate and representative of the evolving data, without the need for constant, full recalculations.

One key advantage of **VCP** is its adaptability. Unlike other methods with fixed thresholds, **VCP** accounts for the natural spread of clusters dynamically. For example, a 10% variance shift in a widely distributed cluster and a 10% shift in a tightly packed cluster are treated with equal significance, maintaining consistent and reliable performance across different data distributions. Ultimately, this approach assures that both sparse and dense clusters are treated fairly and efficiently, maintaining high clustering accuracy over time.

B. Algorithm design

Initially, it is appropriate to assume that a set of **N** points are correctly clustered into **n** clusters using existing methods. As such, we can proceed with

Step 1 - Store the initial centroid locations $\overline{C^0}$,

followed by the **Iterative Algorithm:**

Step 2 - A new point $\boldsymbol{P_{N+1}}$ enters the data stream
Step 3 - P_{N+1} is assigned to the nearest cluster $\boldsymbol{C_p}$
Step 4 - The centroid of that cluster will shift towards P_{N+1}.
The new location of the cluster centroid $\overline{C_P'}$ is given by:

$$\frac{(|C_P| * \overline{C_P}) + P_{N+1}}{N+1}$$

Step 5 - Calculate the **Variance Change Proportion (VCP):** measure of how much clusters have moved as a proportion of the variance of each cluster.

$$VCP_i = \frac{||\overline{C_i^0} - \overline{C_i'}||^2}{Var(C_i)} \text{ is the } VCP \text{ for } i^{th} \text{ cluster.}$$

and

$$VCP = \sum_{i=1}^{n}(VCP_i) \text{ is the } VCP \text{ for all clusters.}$$

Step 6 - If the $VCP \geq$ some threshold, perform a full re-clustering. Update the initial centroid locations $\overline{C^0}$

The algorithm then repeats **steps 2-6** while the stream of new points is being introduced into the system.

C. Efficiency and Scalability

Our algorithm is highly scalable due to its constant-time incremental additions. While reclustering is ultimately unavoidable for maintaining meaningful cluster accuracy, the worst-case time complexity remains equivalent to that of traditional k-means. However, in most scenarios, the average and best-case complexities are significantly better, something that was lacking in many of the existing works for real-time clustering.

The algorithm also performs best when data points contain fewer outliers. Since each new point affects the centroids of existing clusters, points that do not cause significant shifts allow for more incremental additions without triggering a reclustering event. For example, if incoming data points fall within the bounding box of existing clusters, they are easily assimilated, minimizing centroid shifts and delaying computationally expensive calculations. Furthermore, even in cases where outliers dominate the data, the algorithm incrementally processes as many points as possible until the Variance Change Proportion (VCP) threshold is reached. At that stage, reclustering ensures the clusters remain accurate and meaningful.

Additionally, our algorithm maintains its efficiency regardless of the sparsity or density of clusters. Adding points to a sparse cluster is handled with the same computational efficiency as adding points to denser clusters. This is because our Variance Change Proportion (VCP) metric accounts for the spread of each cluster. Centroid shifts are evaluated relative to the cluster's original variance, ensuring that reclustering events are not prematurely triggered by differences in density. This adaptability makes our algorithm highly versatile and suitable for a wide range of data distributions.

IV. Analysis

A. Experiments

To test our approach, we generated a synthetic dataset consisting of 1500 initial points and 700 additional points. The initial points consist of two sets of points from the 2-dimensional normal distribution. The first set consists of 500 points and comes from a distribution with a mean at (15000, 3000) and standard deviation of (2000, 5000). The second set has 1000 points and comes from a distribution with a mean of (10000, 20000) and standard deviation of (7400, 4400). The dataset can be seen in **Figure 1** in the appendix.

The additional points were also generated from two different distributions. The first 450 points came from a random walk distribution with a fixed seed, a step size of 1200, and a start location of (20000, 30000). The next 250 points came from a normal distribution with a mean of (6500, 0) and a standard deviation of (7400, 4400). The dataset after the points where added can be seen in **Figure 2**. The orange area is roughly where the points from the random walk

distribution lay, while the new blue points came from the normal distribution.

The distribution parameters were set before any experiments were run. All of the parameters were set to produce a dataset that would be (subjectively) interesting to run clustering experiments on. Reclustering threshold was set to 0.1

The clustering algorithm was taken from a StatQuest YouTube video[14] and implemented in Python 10.13.0 using about 450 lines of code. An important note is that we chose k=4 without using any common methodologies like the elbow-method, and we did not recluster the same dataset multiple times with different starting points to ensure an optimal clustering. K=4 was chosen before the data was generated because it seemed like an easy number to work with. All the code for clustering was manually written without any third party packages. Notable packages that were used include matplotib to make scatter plots and line graphs, numpy to work with numbers and pandas to work with CSVs. Note that none of these packages have clustering capabilities. The entire code is available on our GitHub[15], allowing anyone to replicate our results.

B. Performance Metrics

To evaluate our method, we considered two metrics: total variance of all points and execution speed. The total variance was compared against the variance given by complete reclustering after adding each point, which produces optimal clustering at each step. The total variance of our method was also compared against another approach where we just added each point to the nearest cluster and moved the centroid. Eventually, this led to points assigned to different clusters being mixed together, as one can see in **Figure 3.**

The execution speed of our method was compared to the execution speed of the naïve, but most accurate approach of complete reclustering after adding each point.

C. Experimental Results

After running the experiments, we found that our algorithm performed exceptionally well. After adding the 700 new points, the optimal variance was 51,603,693.2 and the variance of our clustering method was 51,709,558.6. That is a difference of only 0.205%. On average, the variance of our method aligns with the optimal variance at a rate of 99.714%, with the worst-case match rate being 97.239%. The variance in the method that was never reclustered is 68760642.6, a very significant increase. **Figure 5** in the appendix compares the variances of different approaches using a line graph.

This goes to show that our method is very close to the method producing optimal variance. It achieved these results in 0.352 seconds of running time, while the optimal method took 12.698 seconds to run. This is a 36 fold execution speed increase. The difference in performance would be even more dramatic if we reclusted the same points multiple times using different starting centroids. The fact that we only needed to recluster twice is the main contributor to such dramatic performance differences. One might think this indicates that the new points added did not significantly influence the clustering outcome. This is not the case, as made evident by the fact that not reclustering at all leads to a very high variance compared to reclustering every time.

V. Conclusion

A. Summary of Key Findings

We propose a novel approach called Incremental K-Means Clustering Using Variance Change Proportion: A Dynamic Approach for Real-Time Data Integration. We try to strike a balance between accuracy and efficiency. Our method dynamically shifts the centroid when a new point is added. When the centroid shift relative to the variance of the clusters is above a certain threshold, reclustring is triggered.

Running our solution on a synthetic dataset revealed that our approach is not only accurate, but also efficient. The fact that we only needed to recluster twice after adding 700 points that change the shape of the data, all while maintaining very high accuracy, is an indicator that the Variance Change Proportion is the right metric to signal when it’s time to recluster.

B. Limitations and Challenges

The biggest shortcoming of our work is the lack of further evaluation on more diverse and potentially larger datasets. Additionally, we do not know how to optimally set the Variance Change Proportion Threshold, and whether it should even be changed between different datasets at all.

C. Future Work

Further work is needed in evaluating our approach. Testing our method on a set of diverse and large datasets, as well comparing its performance in speed and accuracy against other solutions is required to better evaluate our method.

References

[1] Narita, K., Hochin, T., Hayashi, Y., Nomiya, H. (2020). Improvement of Incremental Hierarchical Clustering Algorithm by Re-insertion. In: Lee, R. (eds) Computational Science/Intelligence and Applied Informatics. CSII 2019. Studies in Computational Intelligence, vol 848. Springer,Cham. https://doi-org.uml.idm.oclc.org/10.1007/978-3-030-25225-0_8

[2] Chakraborty, S., Nagwani, N.K. (2011). Analysis and Study of Incremental K-Means Clustering Algorithm. In: Mantri, A., Nandi, S., Kumar, G., Kumar, S. (eds) High Performance Architecture and Grid Computing. HPAGC 2011. Communications in Computer and Information Science, vol 169. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-22577-2_46

[3] B. Aaron, D. E. Tamir, N. D. Rishe and A. Kandel, "Dynamic Incremental K-means Clustering," *2014 International Conference on Computational Science and Computational Intelligence*, Las Vegas, NV, USA, 2014, pp. 308-313, doi: 10.1109/CSCI.2014.60. https://ieeexplore-ieee-org.uml.idm.oclc.org/document/6822127

[4] Y. Zhang, K. Tangwongsan and S. Tirthapura, "Fast Streaming kk-Means Clustering With Coreset Caching," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 6, pp. 2740-2754, 1 June 2022, doi: 10.1109/TKDE.2020.301874. https://ieeexplore-ieee-org.uml.idm.oclc.org/document/9174834

[5] V. K. Dehariya, S. K. Shrivastava and R. C. Jain, "Clustering of Image Data Set Using K-Means and Fuzzy K-Means Algorithms," *2010 International Conference on Computational Intelligence and Communication Networks*, Bhopal, India, 2010, pp. 386-391, doi: 10.1109/CICN.2010.80. https://ieeexplore-ieee-org.uml.idm.oclc.org/document/5702000

[6] Y. -C. Chang, H. Yang and S. Kong, "Based on Mini Batch K-Means Clustering for Customer Segmentation in E-commerce," *2022 International Conference on Cloud Computing, Big Data and Internet of Things (3CBIT)*, Wuhan, China, 2022, pp. 60-66, doi: 10.1109/3CBIT57391.2022.00021. https://ieeexplore-ieee-org.uml.idm.oclc.org/document/10053363

[7] M. S. Premkumar and S. H. Ganesh, "A Median Based External Initial Centroid Selection Method for K-Means Clustering," 2017 World Congress on Computing and Communication Technologies (WCCCT), Tiruchirappalli, India, 2017, pp. 143-146, doi: 10.1109/WCCCT.2016.42. https://ieeexplore-ieee-org.uml.idm.oclc.org/document/8074511

[8] D. Marutho, S. Hendra Handaka, E. Wijaya and Muljono, "The Determination of Cluster Number at k-Mean Using Elbow Method and Purity Evaluation on Headline News," 2018 International Seminar on Application for Technology of Information and Communication, Semarang, Indonesia, 2018, pp. 533-538, doi: 10.1109/ISEMANTIC.2018.8549751. https://ieeexplore-ieee-org.uml.idm.oclc.org/document/8549751

[9] *S. Na, L. Xumin and G. Yong, "Research on k-means Clustering Algorithm: An Improved k-means Clustering Algorithm," 2010 Third International Symposium on Intelligent Information Technology and Security Informatics, Jian, China, 2010, pp. 63-67, doi: 10.1109/IITSI.2010.74. https://ieeexplore-ieee-org.uml.idm.oclc.org/document/5453745*

[10] C. Liu, R. Hey and W. Wang, "K-AP Clustering Algorithm for Large Scale Dataset," 2011 First International Workshop on Complexity and Data Mining, Nanjing, China, 2011, pp. 87-89, doi: 10.1109/IWCDM.2011.28. https://ieeexplore-ieee-org.uml.idm.oclc.org/document/6128425

[11] I. Pauletic, L. N. Prskalo and M. B. Bakaric, "An Overview of Clustering Models with an Application to Document Clustering," *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, Opatija, Croatia, 2019, pp. 1659-1664, doi: 10.23919/MIPRO.2019.8756868. https://ieeexplore-ieee-org.uml.idm.oclc.org/document/8756868

[12] Zhen Wang and Meng Yang, "A fast clustering algorithm in image segmentation," *2010 2nd International Conference on Computer Engineering and Technology*, Chengdu, 2010, pp. V6-592-V6-594, doi: 10.1109/ICCET.2010.5486041. https://ieeexplore-ieee-org.uml.idm.oclc.org/document/5486041

[13] Vardakas, G., Likas, A. Global k-means++: an effective relaxation of the global k-means clustering algorithm. *Appl Intell* **54**, 8876–8888 (2024). https://doi-org.uml.idm.oclc.org/10.1007/s10489-024-05636-2

[14] J. Starmer, "StatQuest: K-means clustering," *YouTube*. May 23, 2018. [YouTube Video]. Available: https://www.youtube.com/watch?v=4b5d3muPQmA

[15] Eddie Butkaliuk, "GitHub - eddie15teddy/DMproj," *GitHub*, 2024. https://github.com/eddie15teddy/DMproj (accessed Dec. 09, 2024).

Appendix A

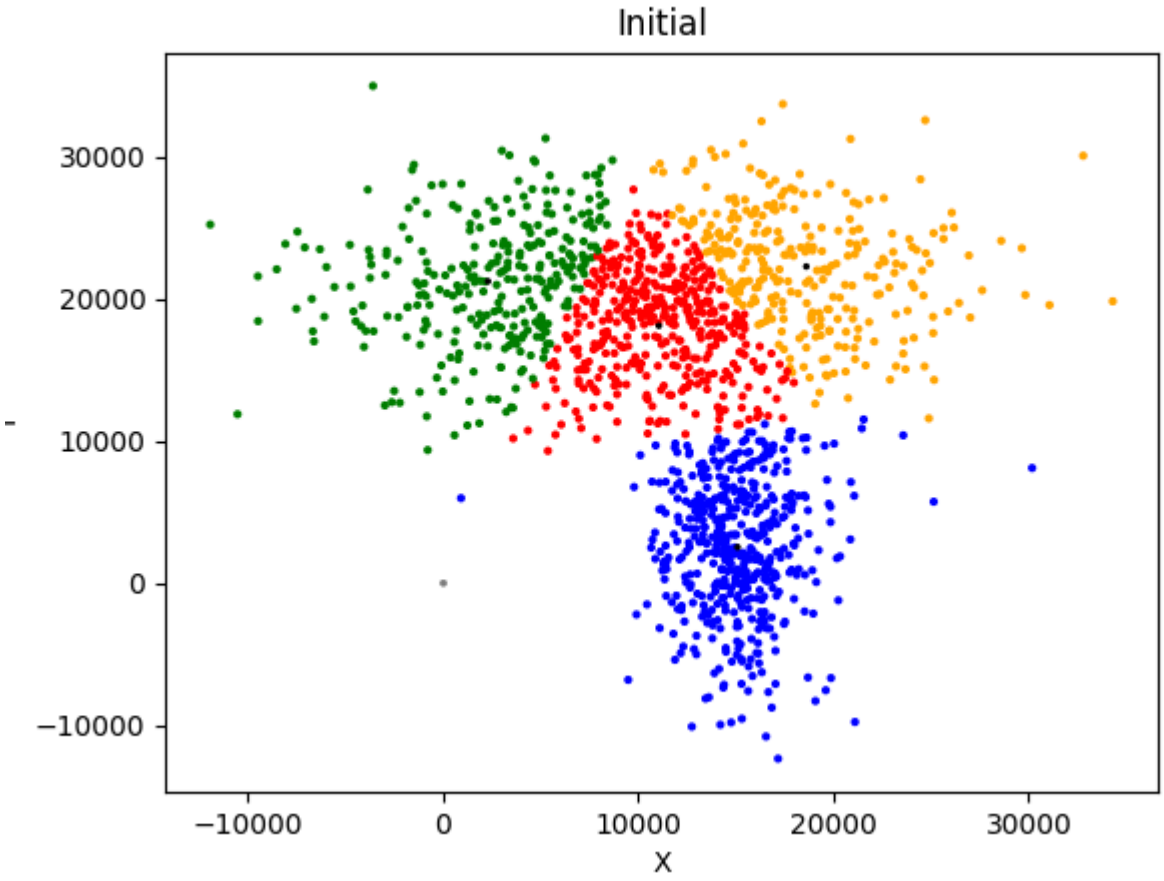


Figure 1: The initial test dataset clustered.

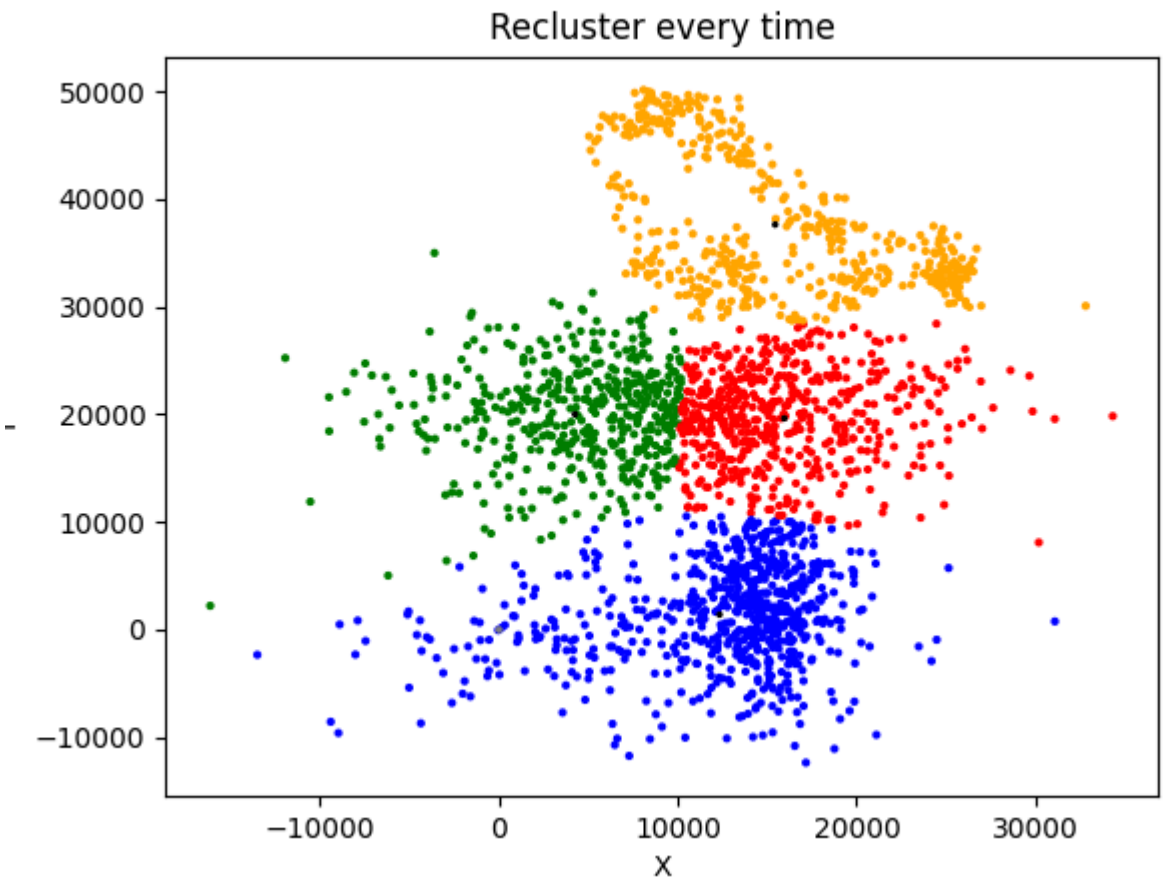


Figure 2: The test dataset properly clustered after the additional points were added.

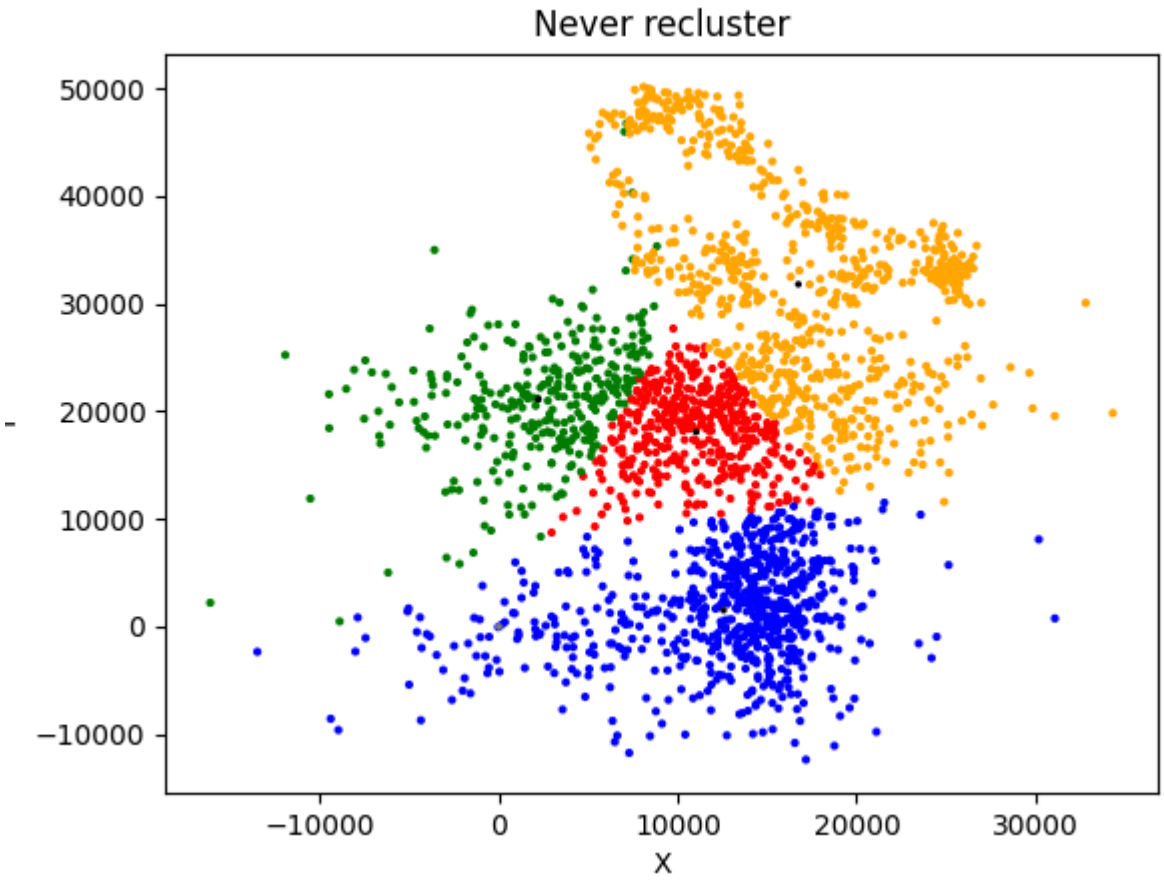


Figure 3: Each point was added to the nearest cluster, and the centroid was moved. Points previously assigned to a cluster were never reassigned.

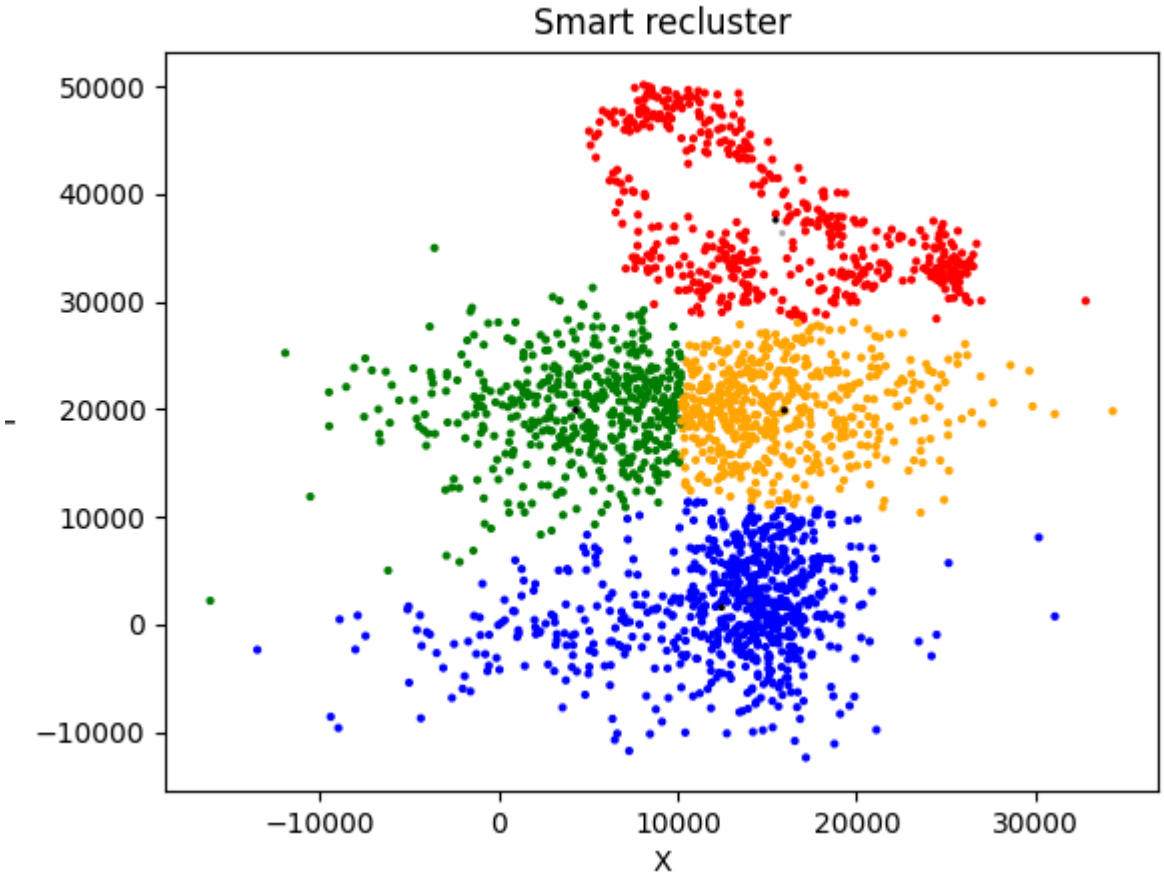
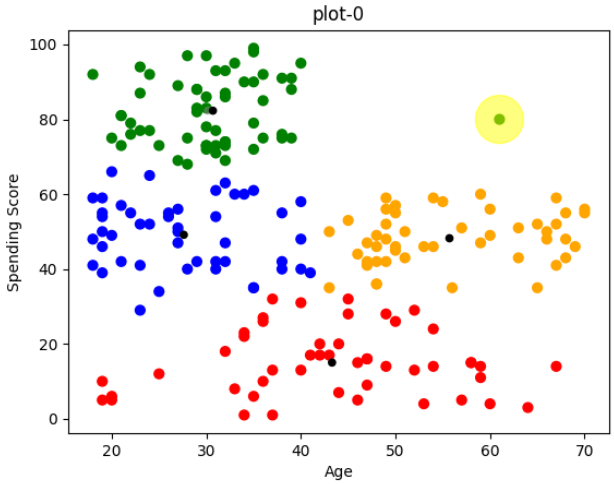
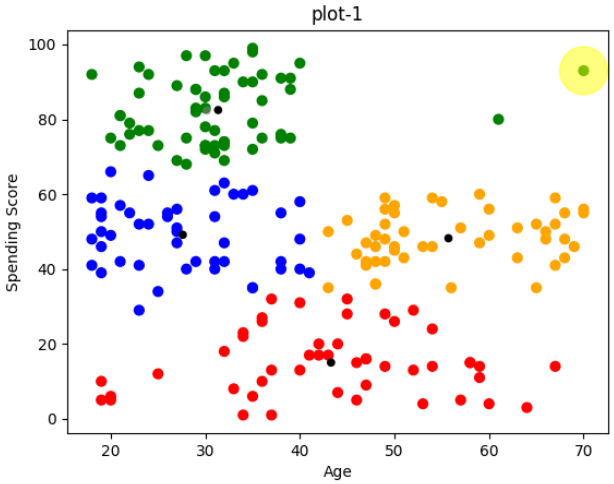
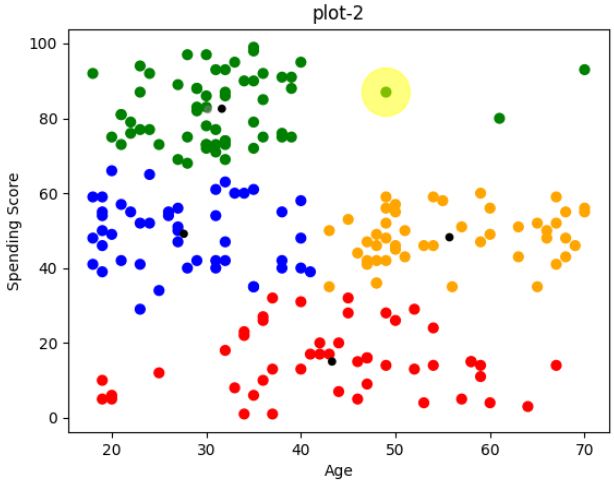
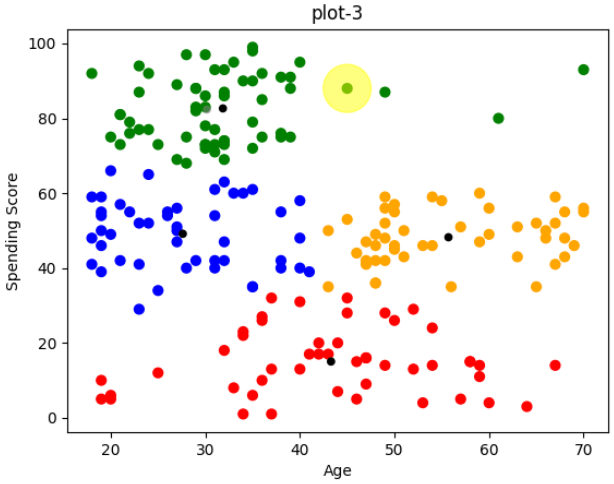


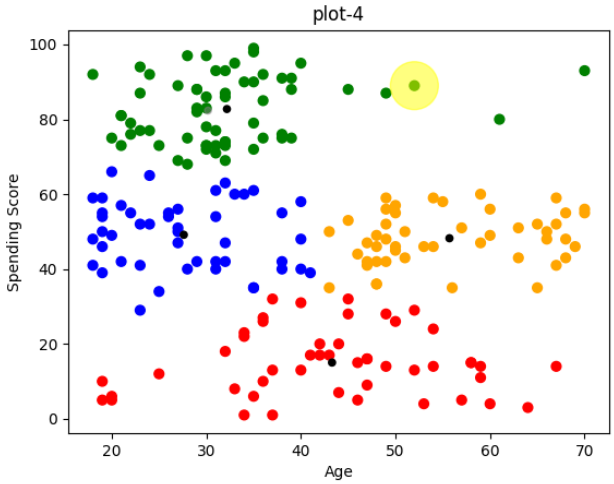
Figure 4: Reclustered using our proposed method.

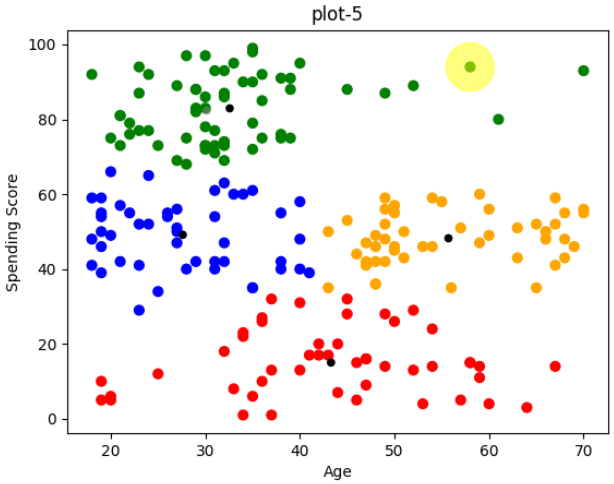

plot-0
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

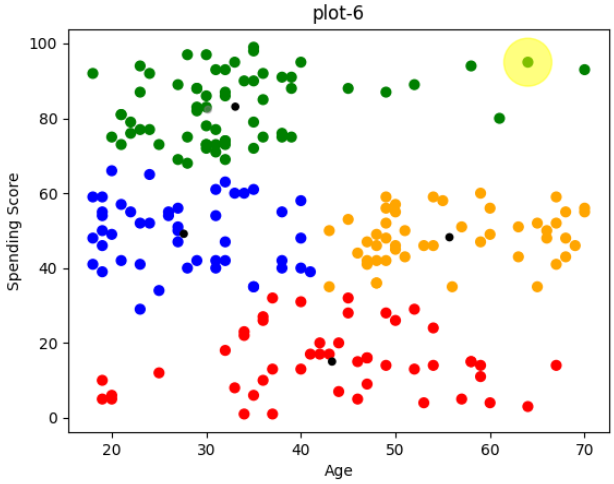

plot-1
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

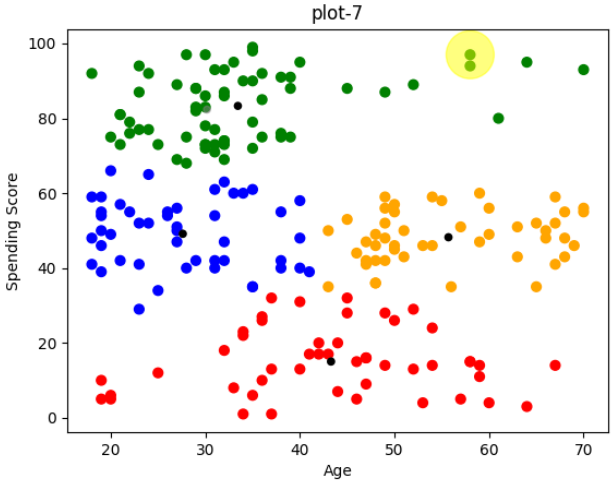

plot-2
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

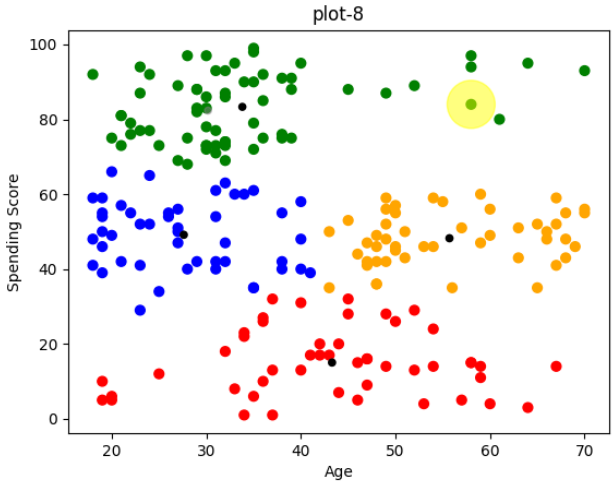

plot-3
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

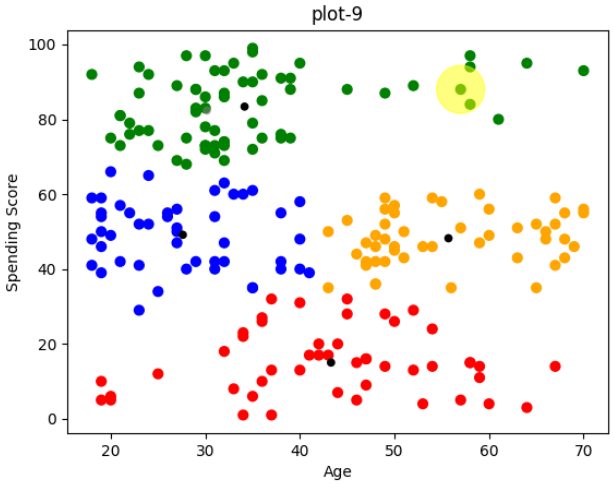

plot-4
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

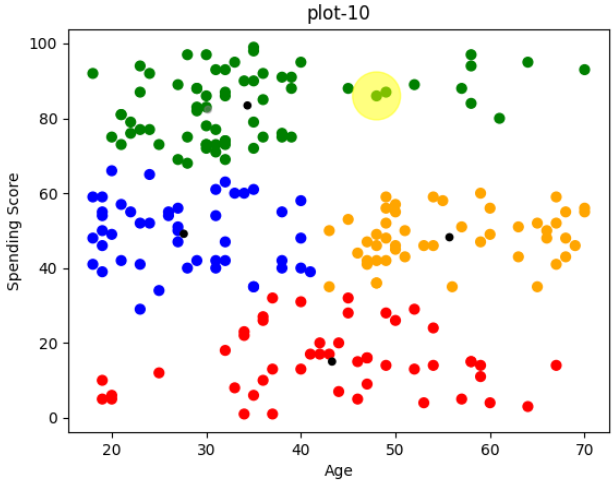

plot-5
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

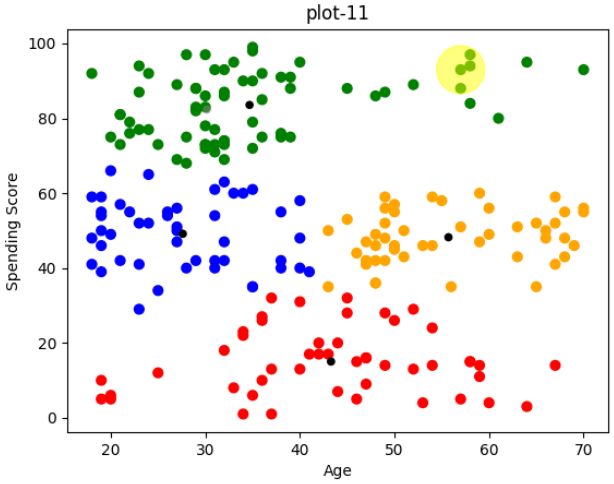

plot-6
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

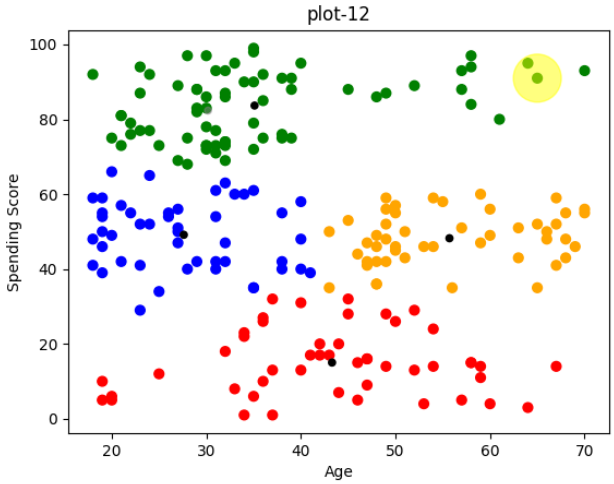

plot-7
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

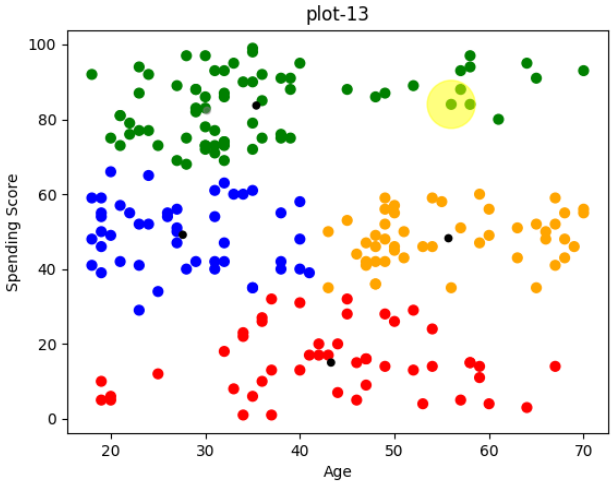

plot-8
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

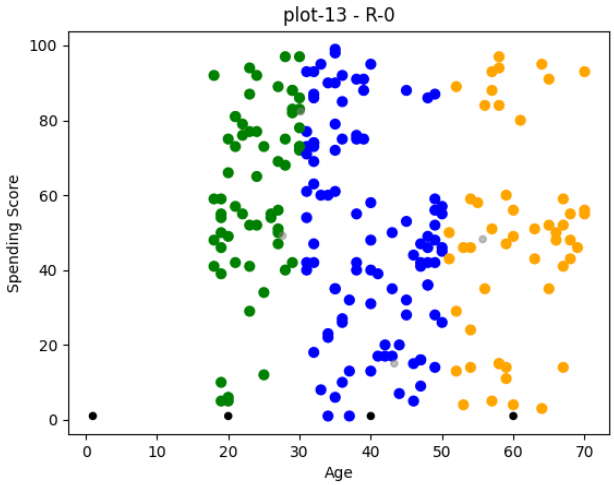

plot-9
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

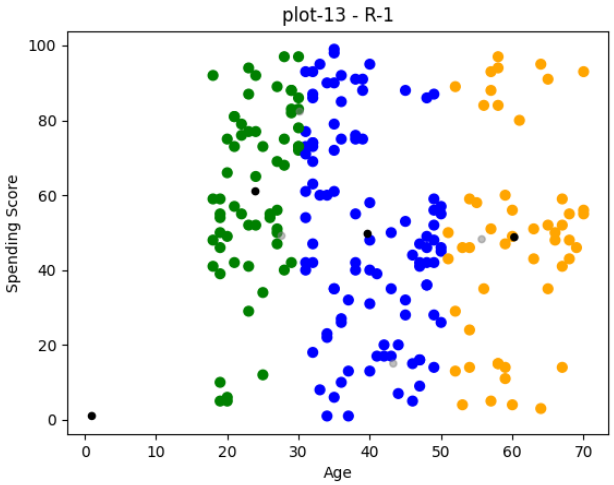

plot-10
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

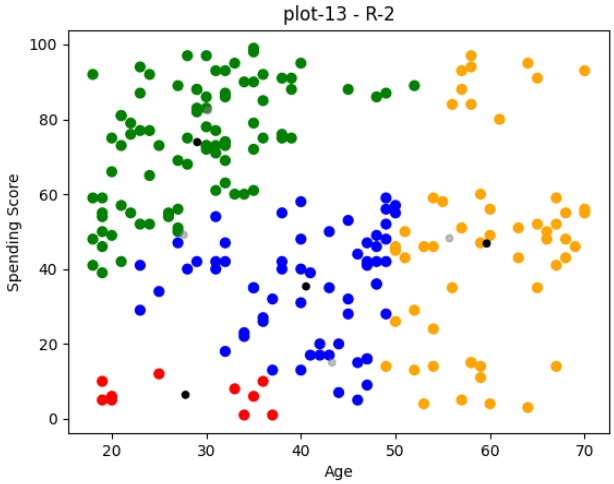

plot-11
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

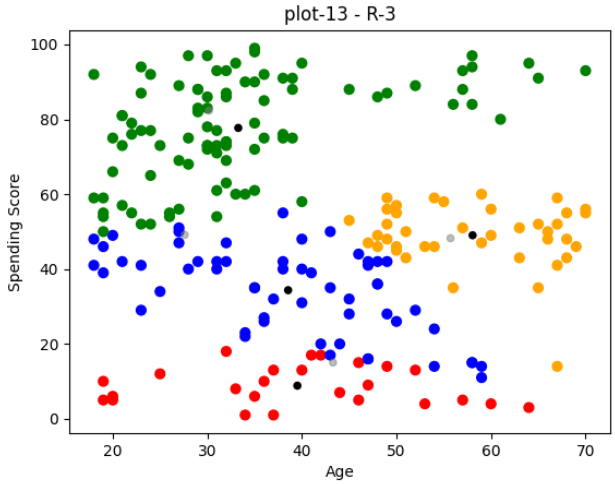

plot-12
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

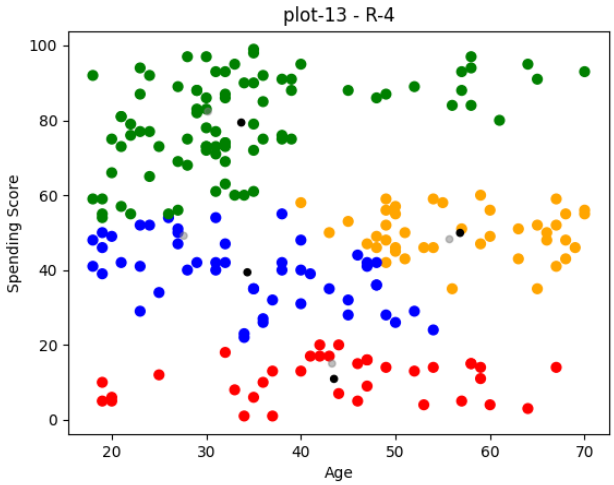

plot-13
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

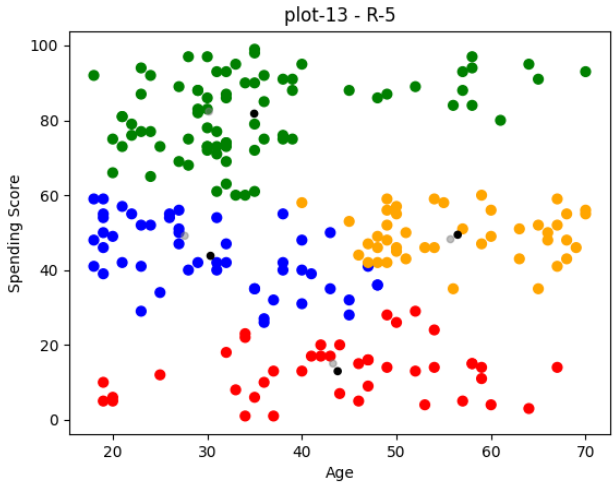

plot-13 - R-0
Spending Score
Age
0
20
40
60
80
100
0
10
20
30
40
50
60
70

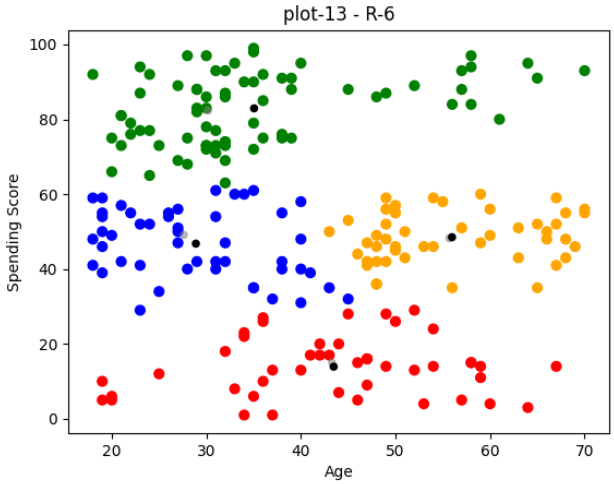

plot-13 - R-1
Spending Score
Age
0
20
40
60
80
100
0
10
20
30
40
50
60
70

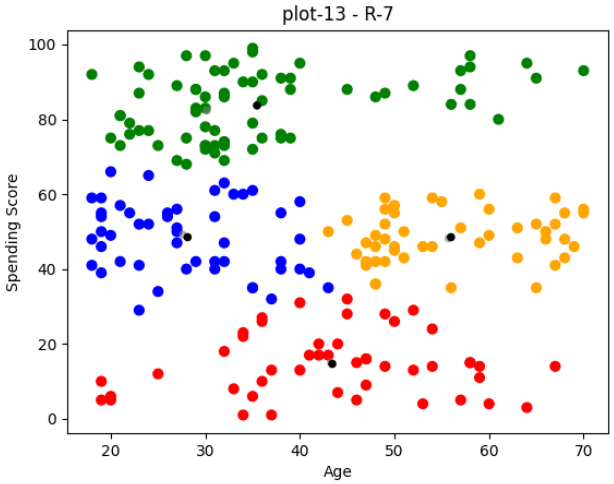

plot-13 - R-2
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

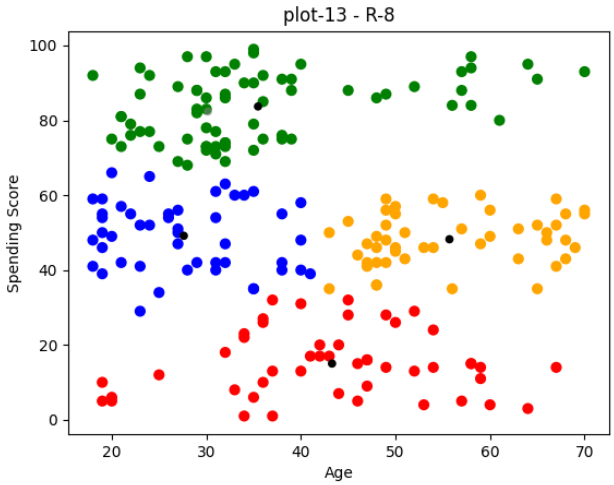

plot-13 - R-3
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

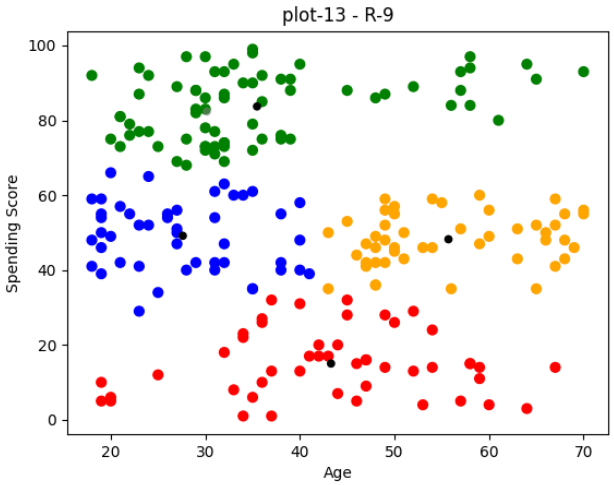

plot-13 - R-4
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

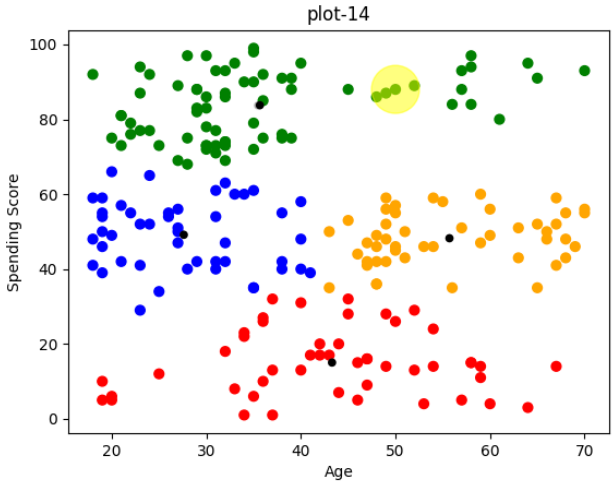

plot-13 - R-5
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

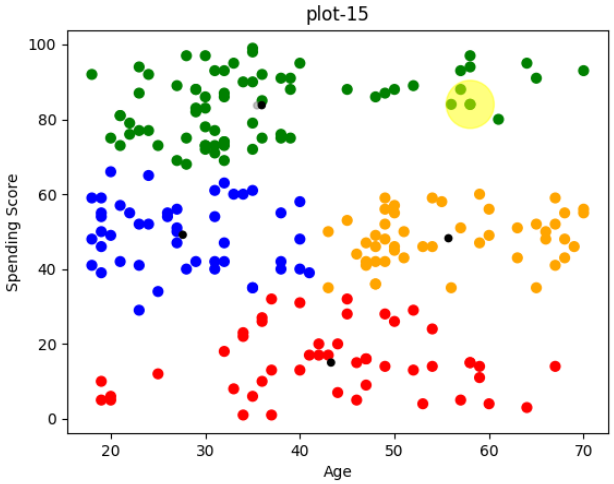

plot-13 - R-6
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

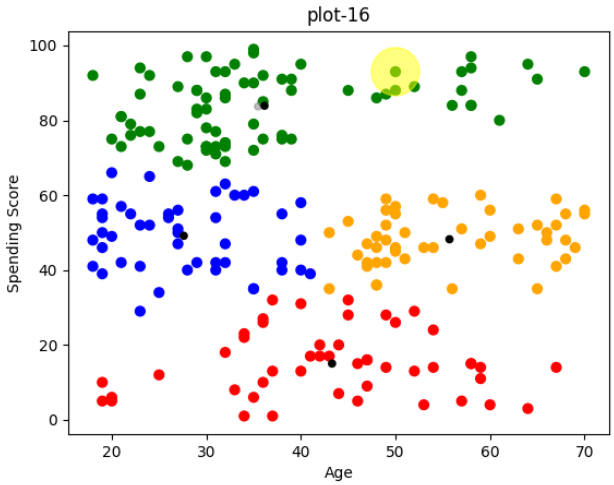

plot-13 - R-7
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

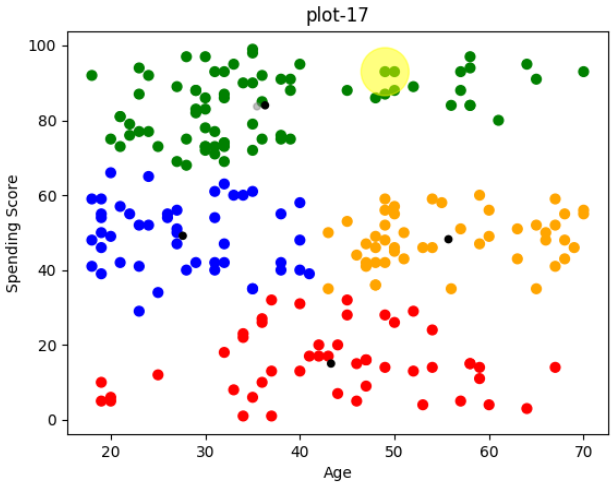

plot-13 - R-8
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

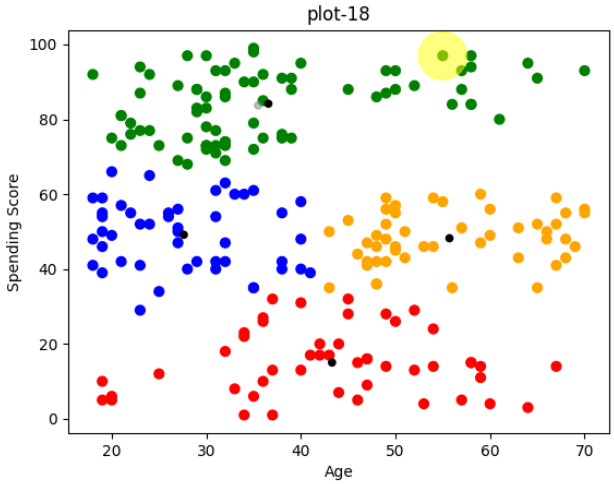

plot-13 - R-9
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

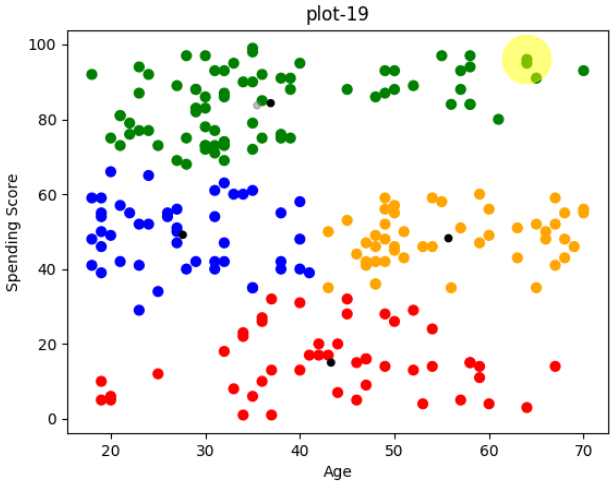

plot-14
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

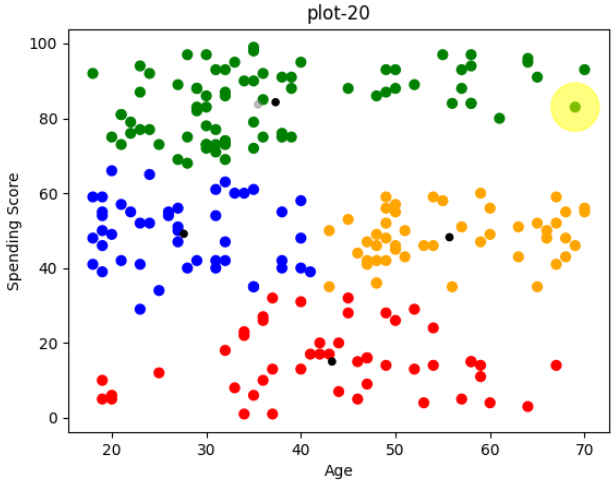

plot-15
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

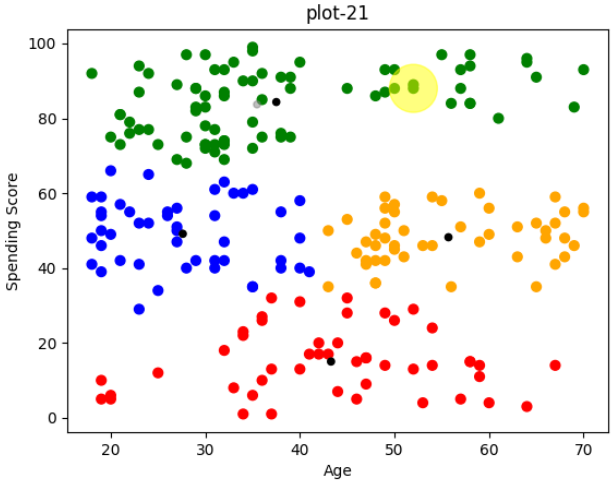

plot-16
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

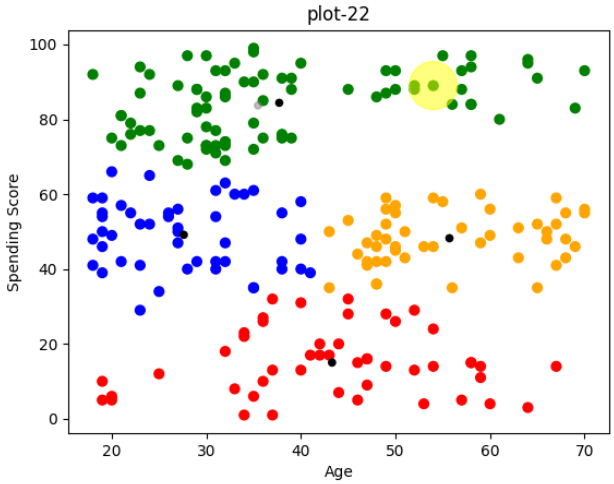

plot-17
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

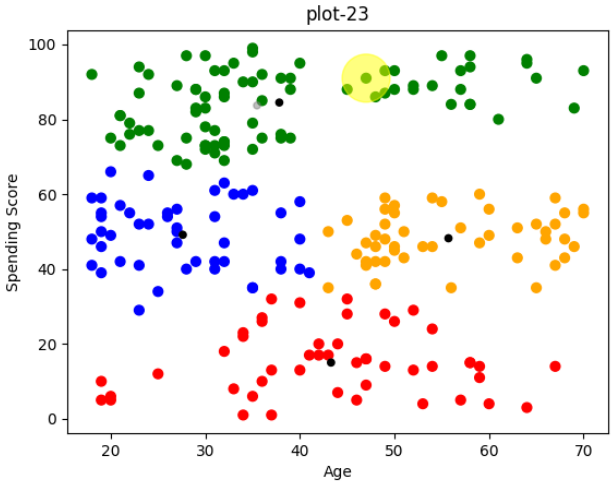

plot-18
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

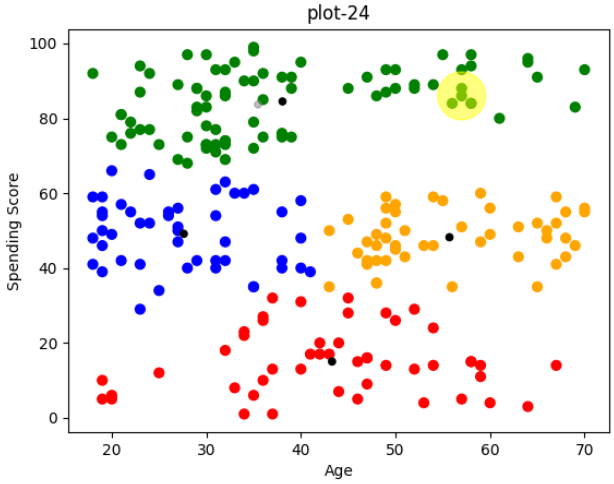

plot-19
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

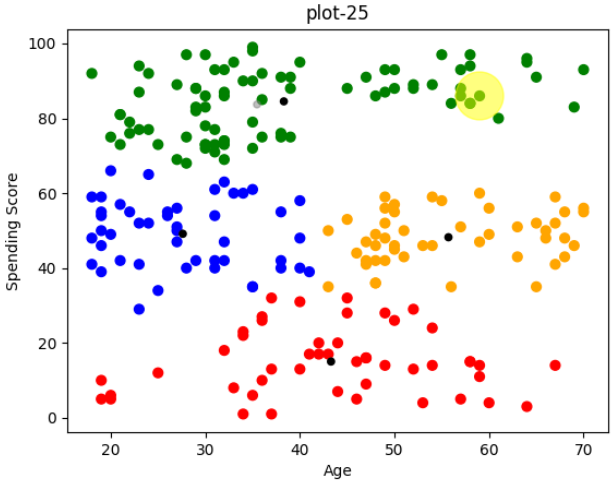

plot-20
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

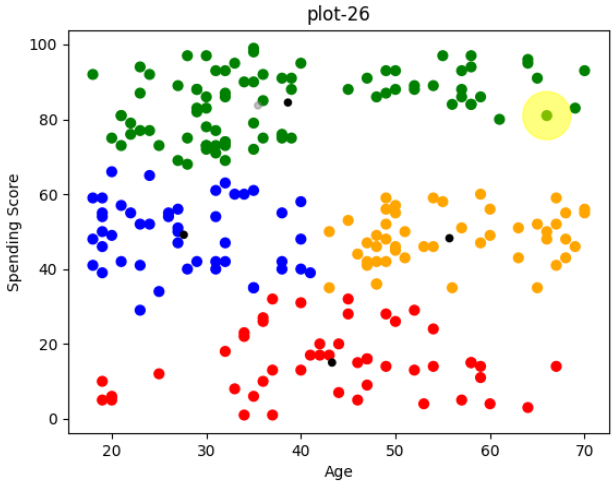

plot-21
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

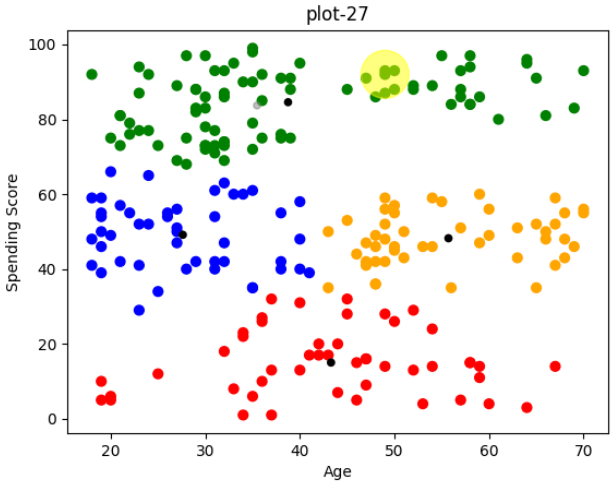

plot-22
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

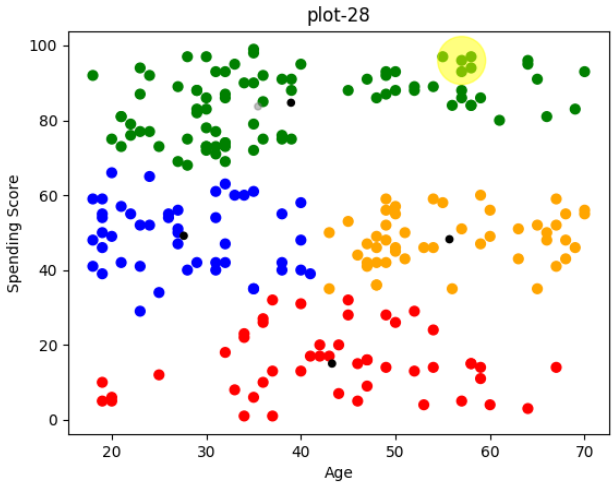

plot-23
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

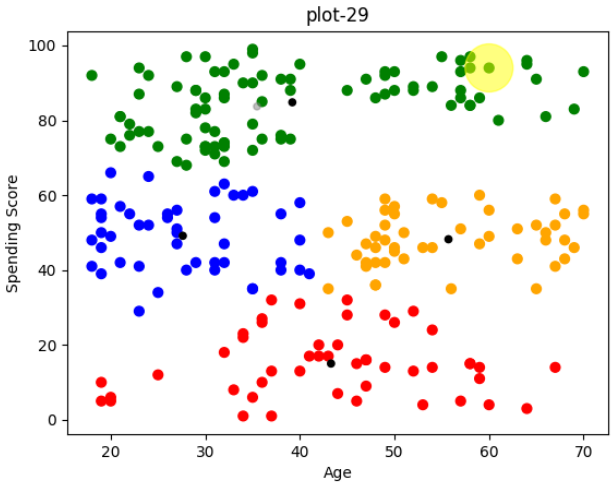

plot-24
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

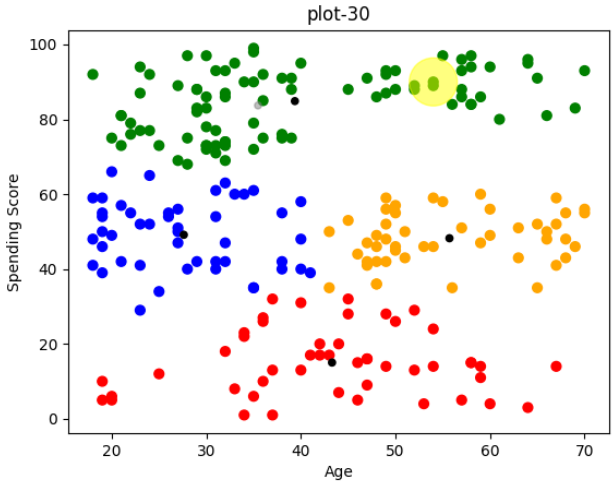

plot-25
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

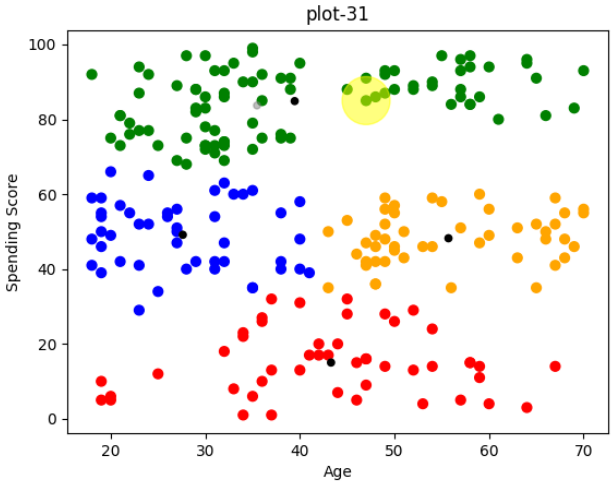

plot-26
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

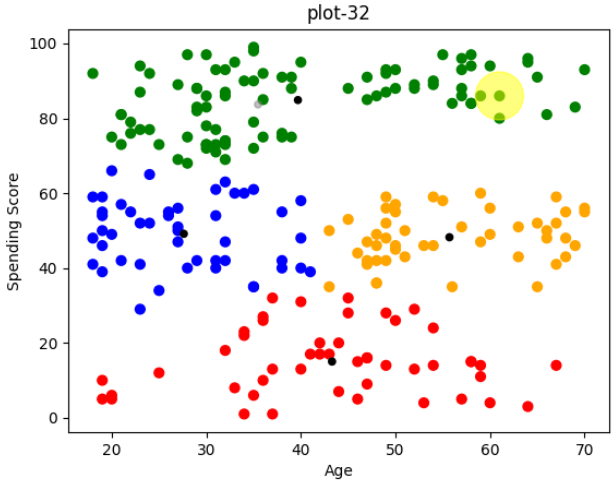

plot-27
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

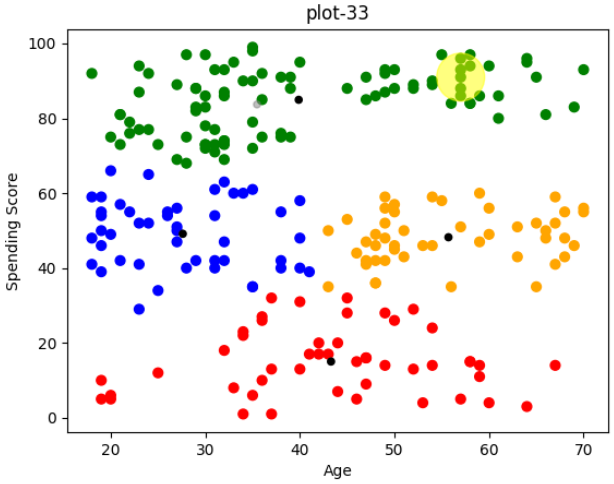

plot-28
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

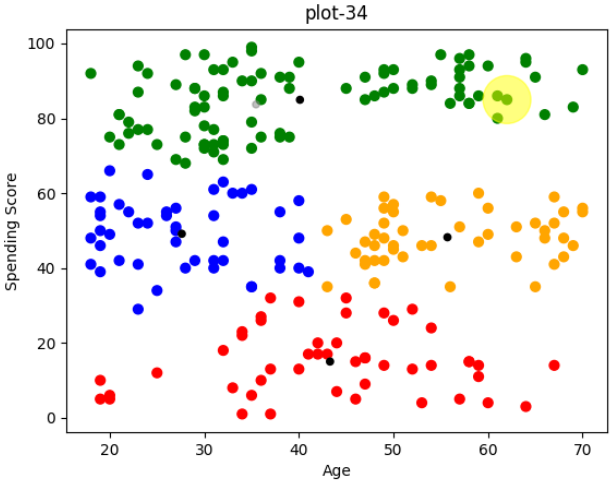

plot-29
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

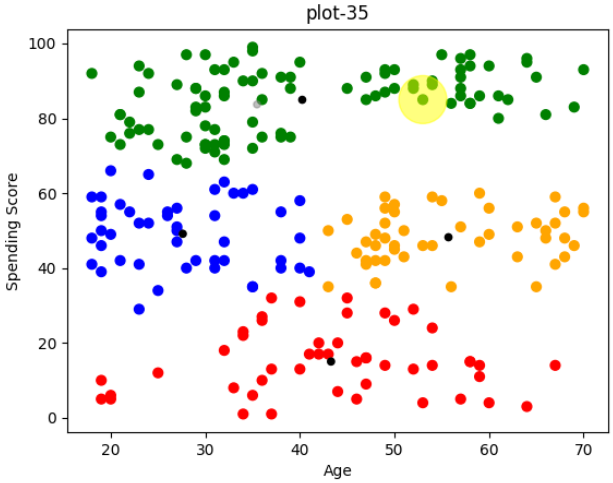

plot-30
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

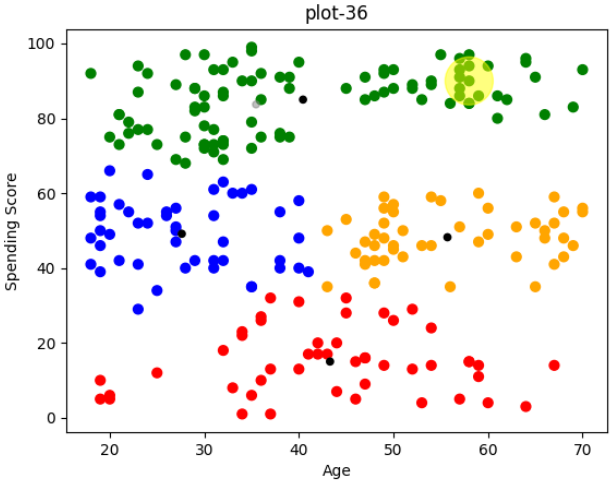

plot-31
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

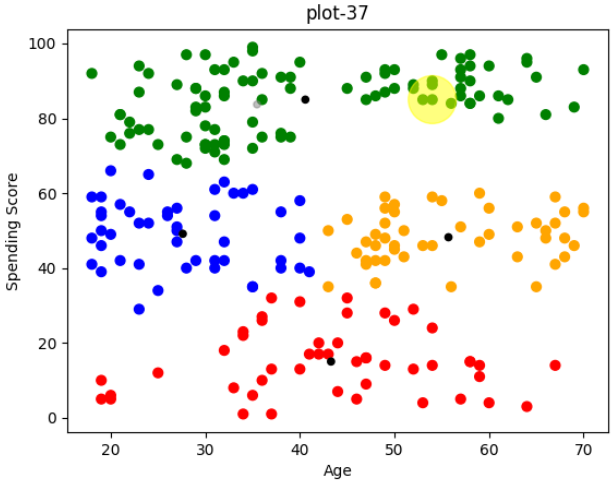

plot-32
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

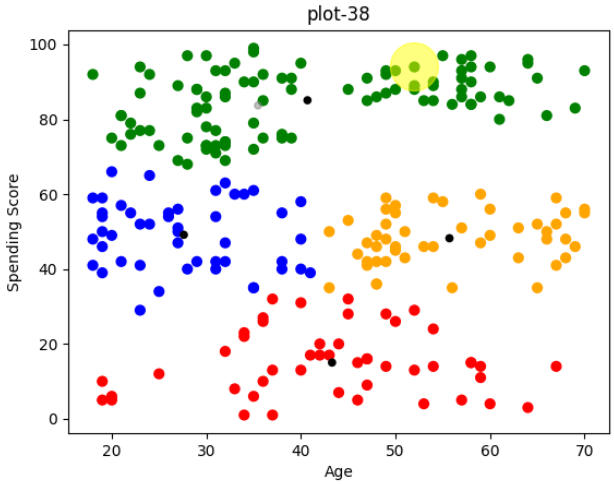

plot-33
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

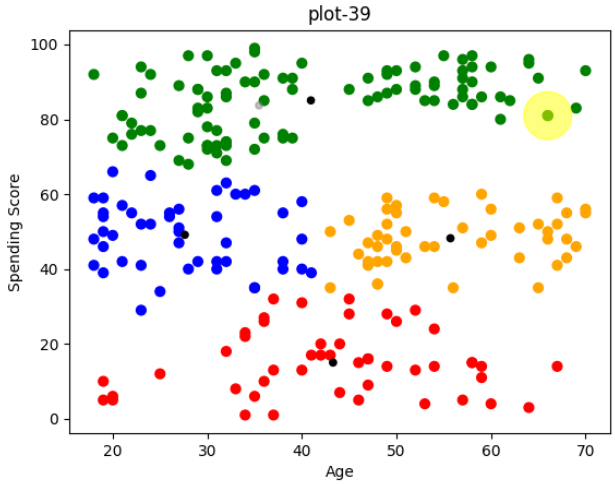

plot-34
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

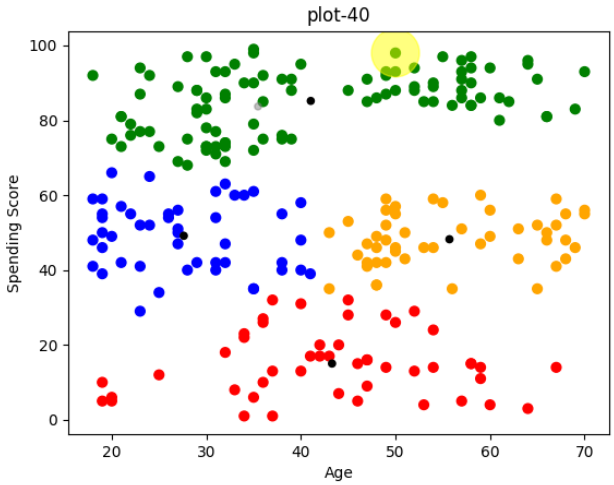

plot-35
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

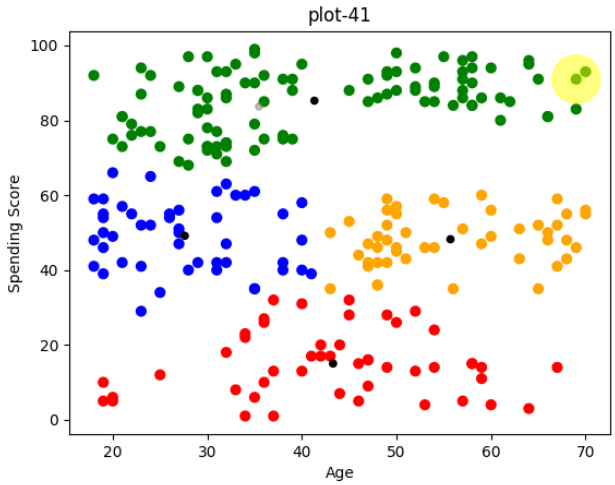

plot-36
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

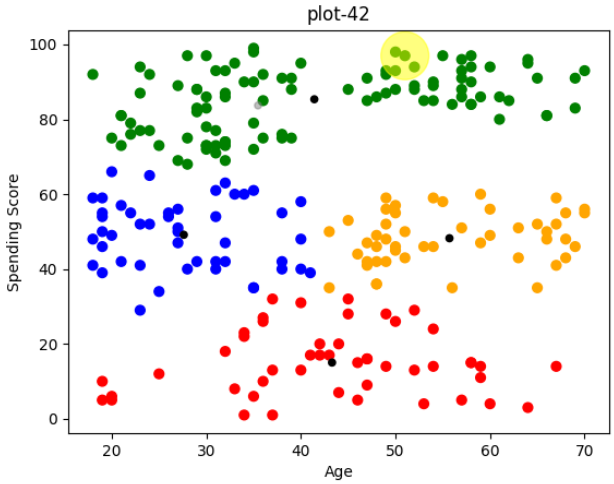

plot-37
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

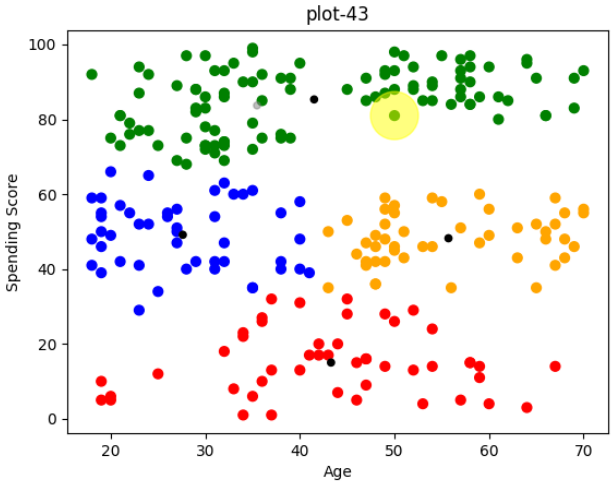

plot-38
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

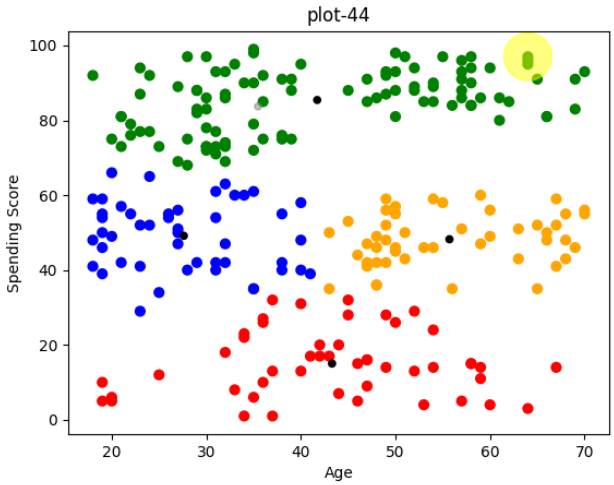

plot-39
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

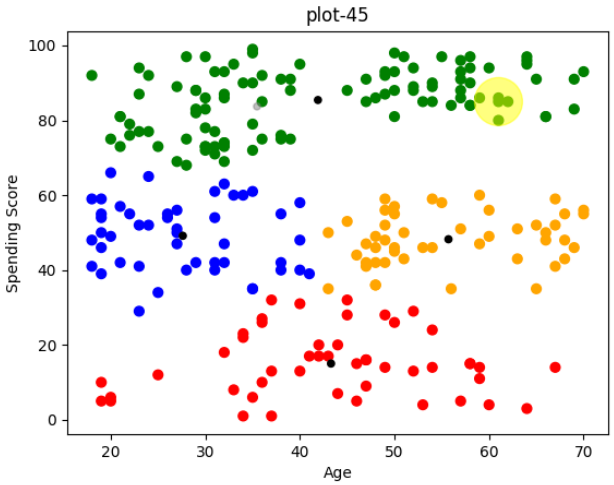

plot-40
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

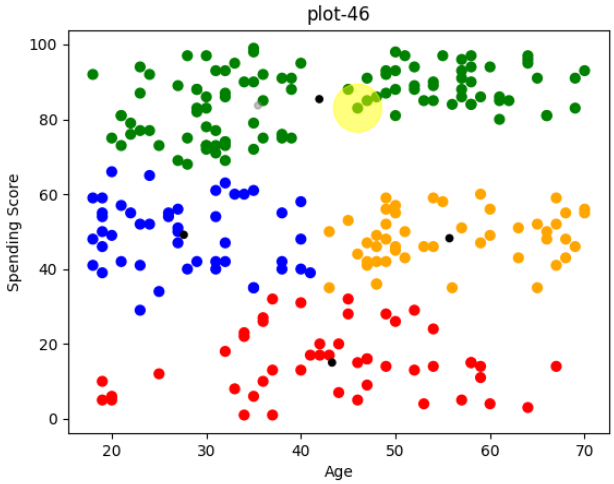

plot-41
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

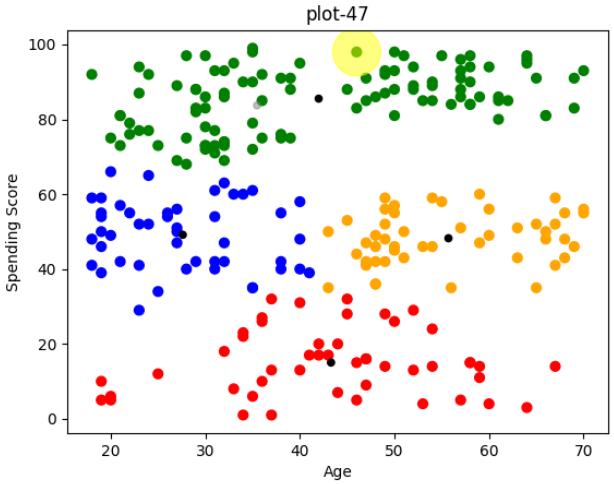

plot-42
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

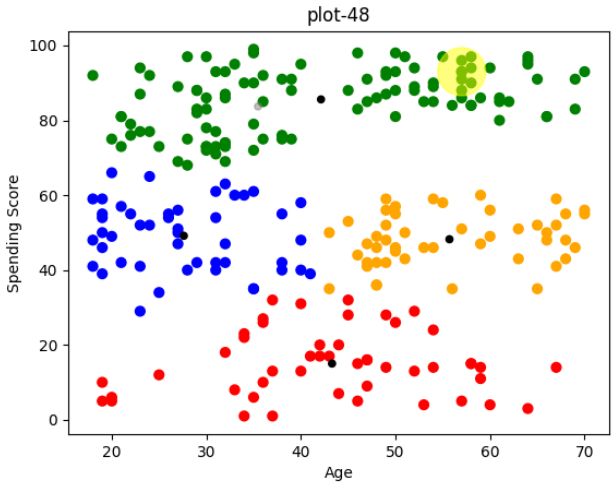

plot-43
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

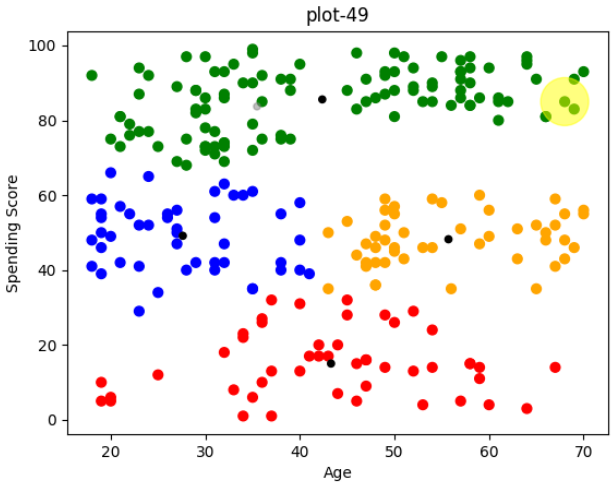

plot-44
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

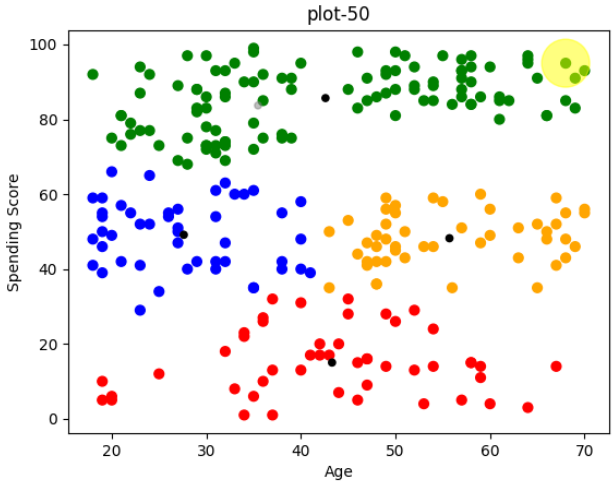

plot-45
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

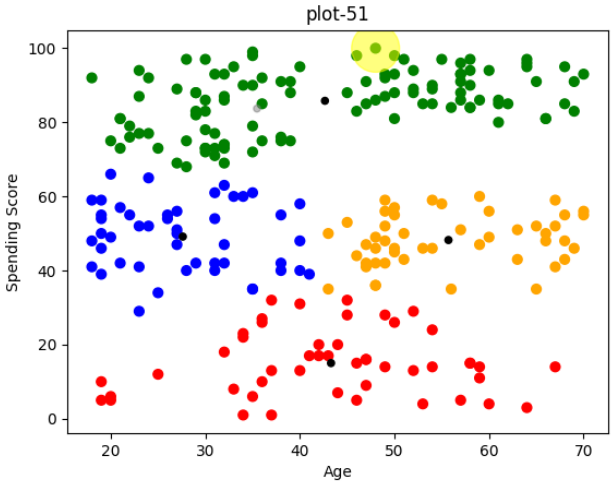

plot-46
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

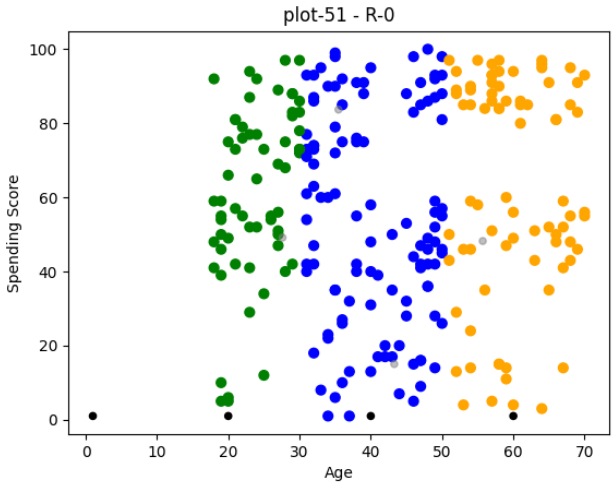

plot-47
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

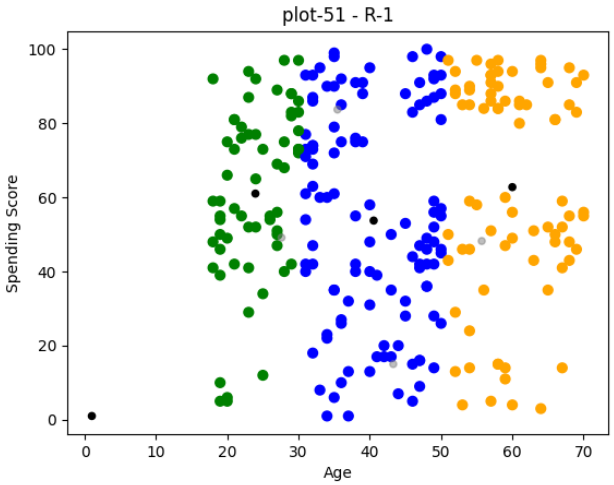

plot-48
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

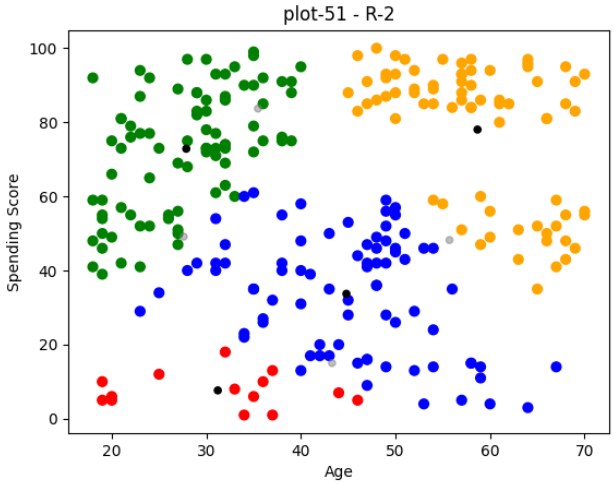

plot-49
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

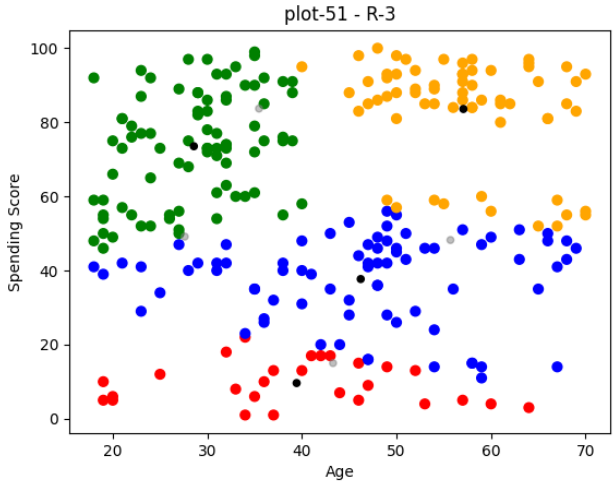

plot-50
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

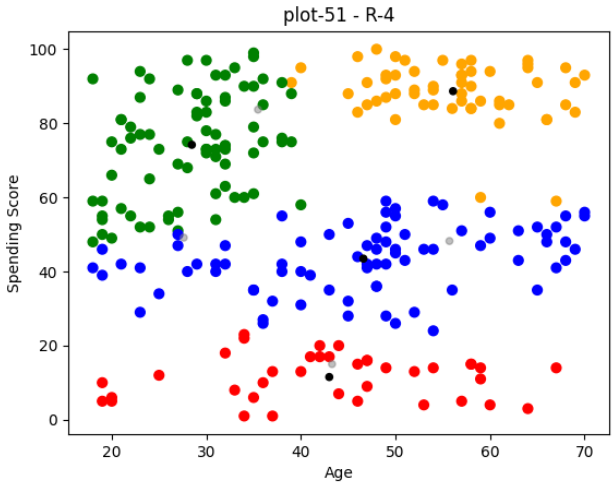

plot-51
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

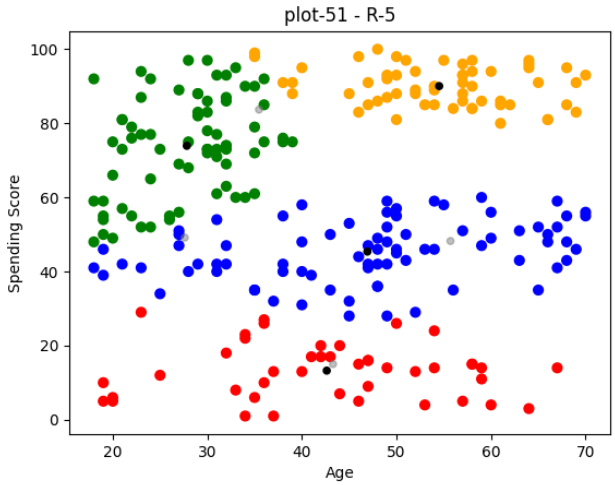

plot-51 - R-0
Spending Score
Age
0
20
40
60
80
100
0
10
20
30
40
50
60
70

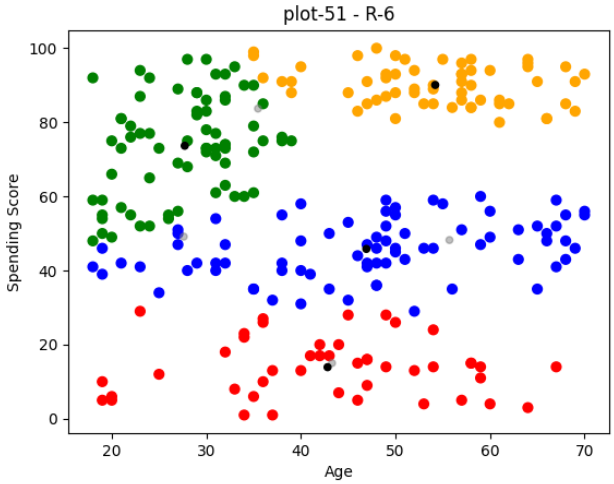

plot-51 - R-1
Spending Score
Age
0
20
40
60
80
100
0
10
20
30
40
50
60
70

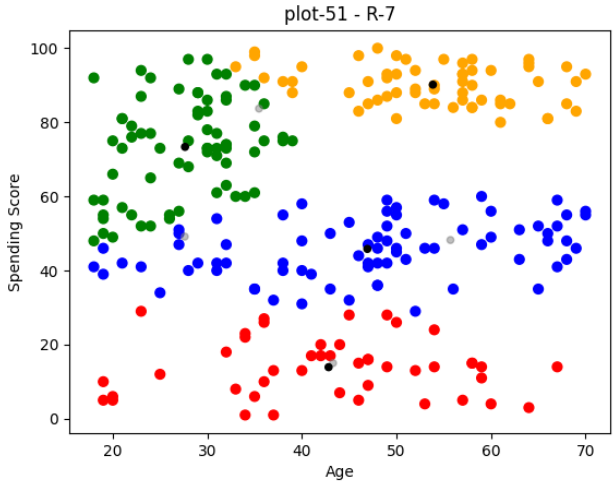

plot-51 - R-2
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

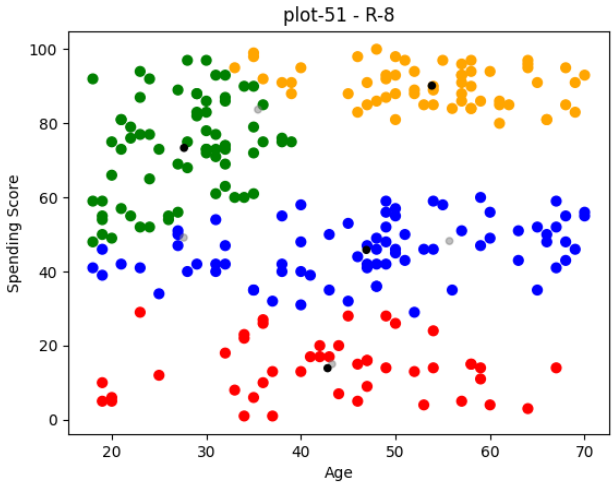

plot-51 - R-3
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

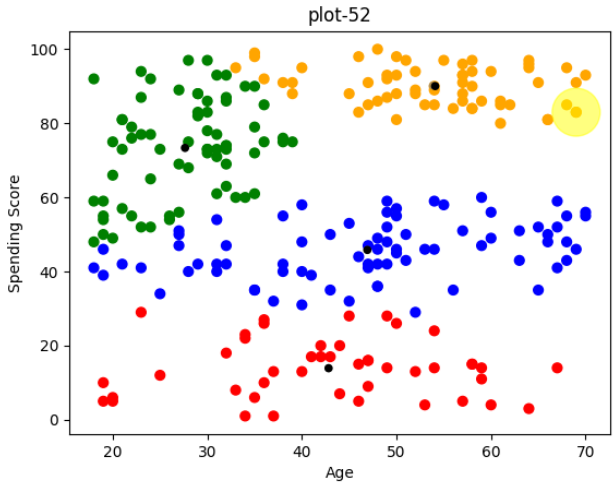

plot-51 - R-4
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

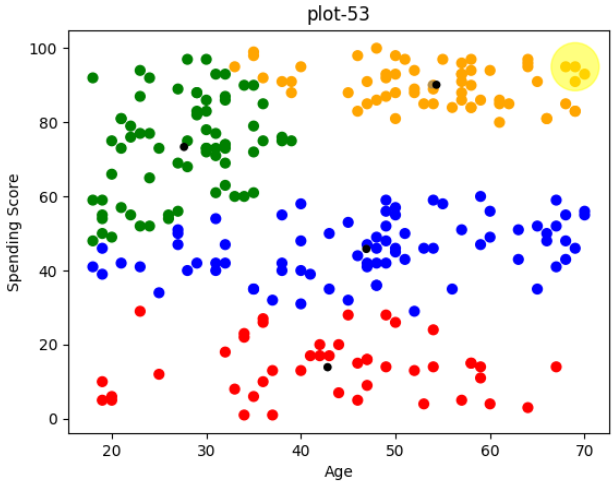

plot-51 - R-5
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

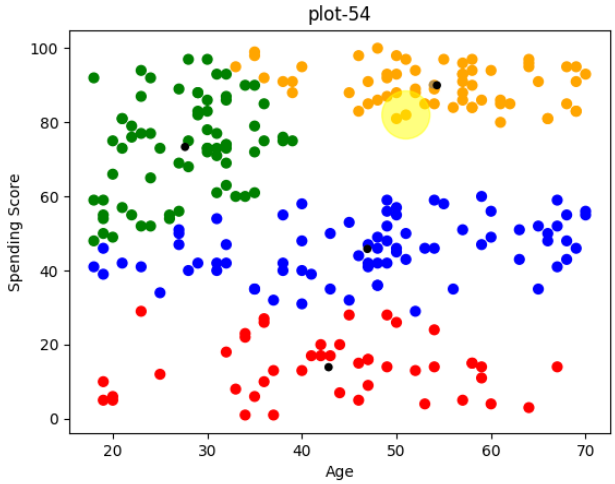

plot-51 - R-6
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70


plot-51 - R-7
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70


plot-51 - R-8
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70


plot-52
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70


plot-53
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70


plot-54
Spending Score
Age
0
20
40
60
80
100
20
30
40
50
60
70

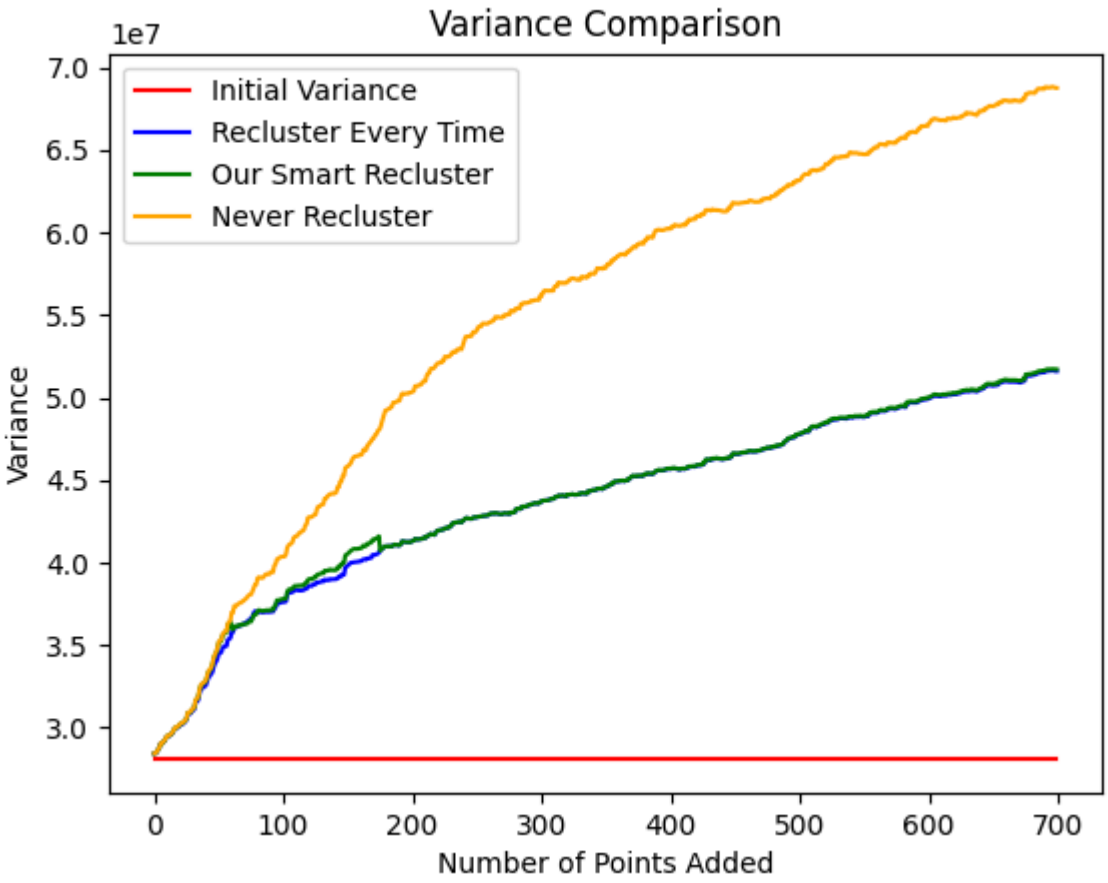


Figure 5: Variance comparison.