University of Cape Town

Department of Statistical Sciences

Analytics

Assignment 3

Authors:
Dominique Chetty
Nesan Naidoo

Student Numbers:
CHTDOM001
NDXNES005

May 29, 2025

Introduction

Clustering is an unsupervised learning method that finds natural groups in data based on similar characteristics. Real-world data often does not follow simple patterns, so clustering needs an iterative process. This means trying different approaches to understand the data better.

In this report, we examined a synthetic dataset. Furthermore, this report goes through the different stages of cluster analysis workflow namely exploratory data analysis, hyper-parameter tuning and further analysis of our best cluster assignment. Additionally, this report details two clustering algorithms: K-means and K-medoids, to explore their relative strengths and weaknesses and understand how they perform under different data conditions.

Exploratory Data Analysis

Data Description

Our data set consists of 2 numeric variables, V_1 and V_2. When assessing our data, we found that there were no missing values, no duplicate values and no infinite values. Therefore, no extra data cleaning process was required of us.

The "cleaned" dataset that we will be using for this report consists of 5000 observations for each of the 2 variables. Table 1 displays the summary statistics of minimum, maximum and the quartiles for V_1 and V_2 in our dataset.

Table 1: Summary Statistics of Variables V_1 and V_2

Statistic	V_1	V_2
Minimum	89,604	9,597
First Quartile (Q1)	370,419	362,699
Median	509,386	494,896
Mean	502,998	497,113
Third Quartile (Q3)	637,848	631,829
Maximum	932,954	977,215

Distance Metric

Based on our data description, we will use Euclidean distance as our distance metric. This is because V_1 and V_2 are both continuous and numeric variables. Moreover, Table 1 shows that V_1 and V_2 have similar scales and ranges. Hence, each variable will contribute approximately equally to the distance calculation. Therefore, Euclidean distance is the most appropriate distance metric for our data. Furthermore, the Euclidean distance metric is simple and intuitive, as the distance measurements correspond with the actual distances.

Exploratory Analysis

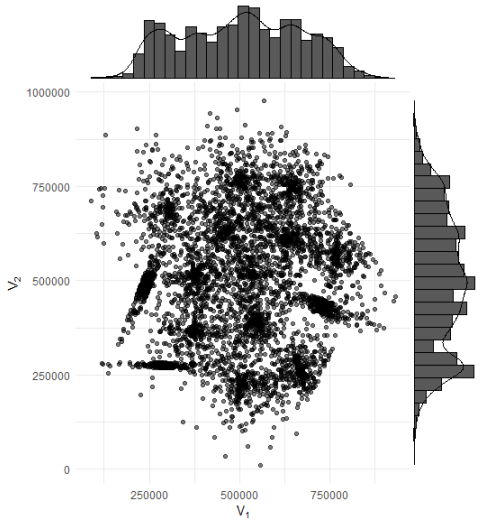


Figure 1: Displaying the univariate and bivariate distributions of V_1 and V_2

From Figure 1, we can see that the univariate distribution of V_1 is multimodal as there are 5 distinct local maxima peaks. This could suggest that there will be approximately 5 ways to cluster the observations based on V_1. Similarly, for the univariate distribution of V_2, we can see that it is also multimodal, as there are 3 distinct peaks, thus suggesting there could be 3 ways to cluster observations based on V_2.

Figure 1 also shows us that the bivariate distribution has a circular shape, with observations clustered with high density around the center point of 500000, with an approximate radius of 250000, for both V_1 and V_2. The high density around this region makes it difficult to cluster the observations, however, Figure 1 displays 15 regions with extremely high density, which could form potential clusters.

The shape of the distributions is important because it will have an effect on the mean. When the distribution is symmetrical, the mean will accurately represent the "center" of the data, which is ideal for K-means clustering. Figure 1 displays the bivariate distribution of our data as being reasonably symmetric. However, since our univariate distributions have multiple peaks, this could indicate that our mean may lie outside of the high-density regions. This would misrepresent our clusters because algorithms like K-means may create artificial clusters or fail to cluster efficiently.

Exploratory Analysis of Distance

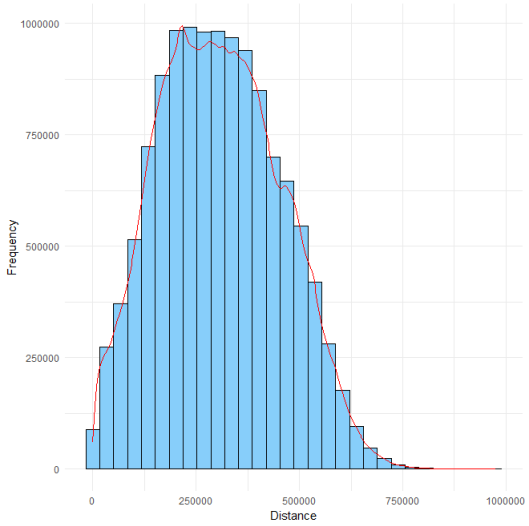


Figure 2: Displaying the distribution of the pairwise distances

Figure 2 shows the distribution of the pairwise Euclidean distances appears to be right-skewed. This suggests that most of the distances are concentrated on the lower end of the distribution, with the presence of some significantly larger distances. The many smaller distances suggest that most data points are close together, likely forming compact and dense clusters. There will likely be many clusters in dense areas, from approximately 200000 to around 4000000. The large distances in the upper tail may represent outliers.

This impacts clustering since K-means minimizes squared euclidean distances, which will make larger distances (outliers) dominate. Therefore, centroids will be biased towards outliers. Dense regions may also be incorrectly split to accommodate outliers. However, K-medoids uses the actual data points instead of the mean. It minimises the absolute distance so outliers will have less influence. Therefore, the right-skewness will not have an impact on K-medoids clustering.

Outlier Identification

To identify 10 outliers, we have calculated the euclidean distance from the median for all observations and have labeled the outliers as the observations with the highest euclidean distance from the median. We chose the median because it is robust to skewness. Table 2 shows the 10 observations with the greatest distance from the median, and Figure 14 in the Appendix shows a visual representation of the outliers. Subsequently, we removed those 10 outliers from our dataset to prevent bias in our clustering algorithms and improve cluster separation later on.

Table 2: 10 Outliers by Distance from the Median

Observation	V1	V2	Distance from Median
461	129 640	884 633	544 152.2
340	215 687	902 015	502 000.7
2540	557 874	9597	487 715.3
4722	567 067	977 215	485 755.9
442	89 604	711 268	472 264.1
486	112 396	741 755	467 482.7
501	115 401	744 137	466 202.6
2317	460 464	35 412	462 081.0
535	268 889	886 754	459 773.1
4723	493 270	951 924	457 312.0

Correlation Analysis

Data was adjusted by removing outliers before calculating the correlation between V_1 and V_2 as outliers can distort the true relationship between the variables. Table 3 shows that the correlation between V_1 and V_2 is 0.074. This suggests that there is almost no linear dependence between the variables. Since this is a very weak positive correlation, no additional adjustments are required. This is because V_1 and V_2 carry independent information, so clustering algorithms will treat them as distinct dimensions and euclidean distance calculations will not be biased by correlated features.

Table 3: Correlation Matrix of the cleaned data

	V_1	V_2
V_1	1	0.0740
V_2	0.0740	1

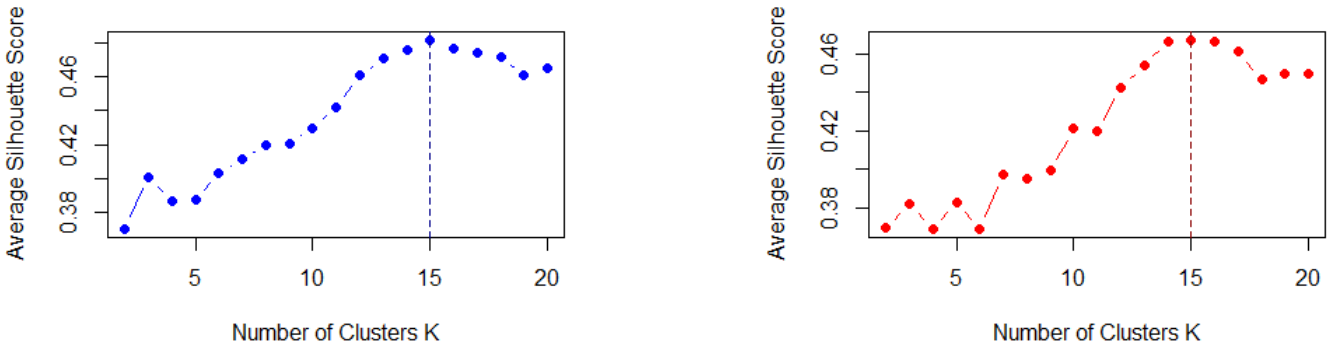
Scaling

Data standardisation is not necessary for this dataset because, according to our data description, both V_1 and V_2 are continuous, numeric variables with similar scales (hundred thousands). Meaning that when euclidean distance is calculated, both variables will have equal influence and no variable will dominate the other. Additionally, since there is very little correlation (0.07) between them, it indicates that there will not be any redundancy.

Hyper-parameter Tuning

Selecting K

Choosing the optimal number of clusters, K^*, in clustering algorithms such as K-means or K-medoids is crucial for generating meaningful and interpretable results. An effective and commonly used diagnostic technique for identifying K^* is the Average Silhouette Score. This technique evaluates clustering quality by measuring how similar each data point is to its own cluster compared to other clusters. The silhouette scores range from -1 (meaning that the data point is in the wrong cluster) to 1 (meaning that the data point is well clustered). By calculating the average silhouette scores for K values within the range of 2 to 20, we will be able to identify the optimal K^*.



(a) K means

(b) K-medoids

Figure 3: Silhouette Method for selecting the optimal K

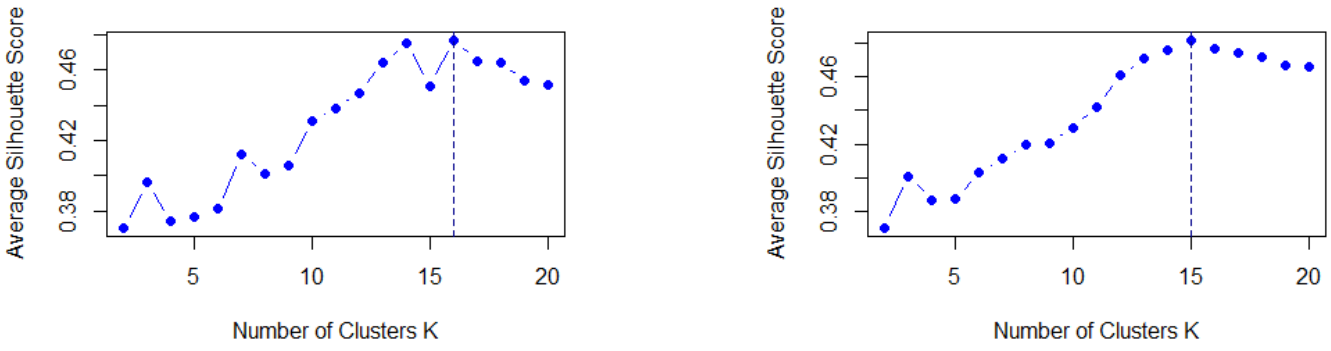
Figure 3a displays the silhouette plot for K-means, which shows an increasing trend in average silhouette scores until it reaches a peak of $K = 15$. This indicates that as the number of clusters increases, the clusters become more

distinguishable and the data points tend to fit better within their assigned cluster. The peak is highlighted by a vertical blue dotted line at $K = 15$, which indicates that the optimal number of clusters using K-means is 15. The scores at $K = 14$ and $K = 16$ are very close to that of $K = 15$, implying they may yield similarly structured clusters. Beyond $K = 15$, the scores gradually decline, suggesting a diminishing cluster quality.

In comparison, 3b displays the silhouette plot for K-medoids, which shows some variability in lower K values, particularly up to $K = 6$. Afterwards, it follows a trend similar to K-means, with average silhouette scores rising and peaking at $K = 15$. The peak is highlighted by a vertical red-dotted line at $K = 15$, which indicates that the optimal number of clusters using K-medoids is 15. Similar to 3a, the silhouette scores at $K = 14$ and $K = 16$ are very close to that of $K = 15$, implying they may yield similarly structured clusters, and after the peak at $k = 15$, there is a gradual decrease in silhouette scores, implying diminishing cluster quality.

Initialisation Sensitivity

The stability and quality of the clustering results for K-means and K-medoids are sensitive to the random selection of initial cluster centroids, which can lead to suboptimal solutions. Therefore, the `nstart` parameter in algorithms such as `kmeans()` and `clara()` allows us to control the number of random initialisations. By running the algorithm multiple times, with different starting points, and selecting the best result, we can reduce the sensitivity to initialisation of the results. Figure 3a was plotted using `nstart=100`.



(a) K-means (nstart=1)

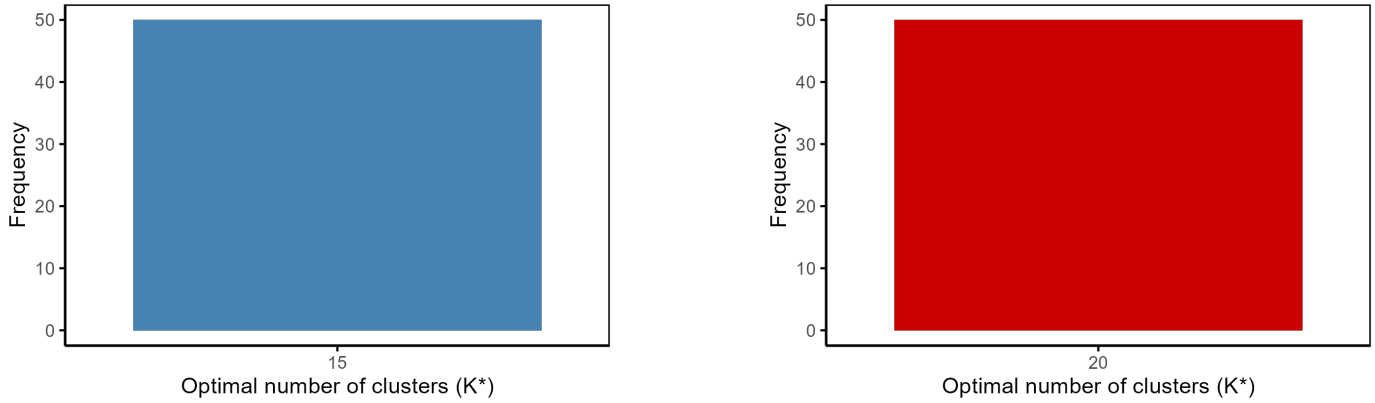
(b) K-means (nstart=100)

Figure 4: Average Silhouette plots for k means(nstart=1 vs 100)

Figures 4a and 4b highlight the significant influence that initialisation has on the stability of clustering. Figure 4a demonstrates that when the algorithm is run once (`nstart=1`), the silhouette score plot is highly volatile, making it difficult to identify clear trends and patterns between the average scores and the number of clusters. In constrast, Figure 4b demonstrates that increasing the number of initialisations to `nstart=100` allows the algorithm to search the solution space more thoroughly, leading to a smoother silhouette score curve. This results in more consistent and reliable clustering outcomes, where the algorithm is less likely to converge to suboptimal solutions.

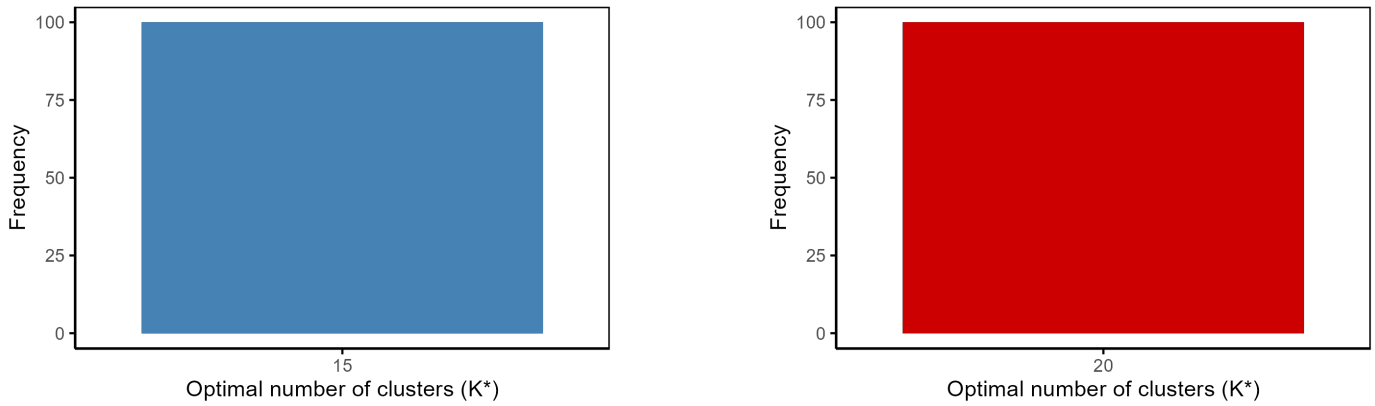
Increasing Initialisations

To efficiently analyse the impact of a large number of initialisations on clustering stability, we parallelize the execution. This helps to speed up the computation by distributing runs across multiple CPU cores. In our analysis, we conducted both 50 and 100 initialisations for each algorithm and summarised the results below.



(a) K-means (b) K-medoids

Figure 5: Distribution of the optimal K across 50 initializations



(a) K-means (b) K-medoids

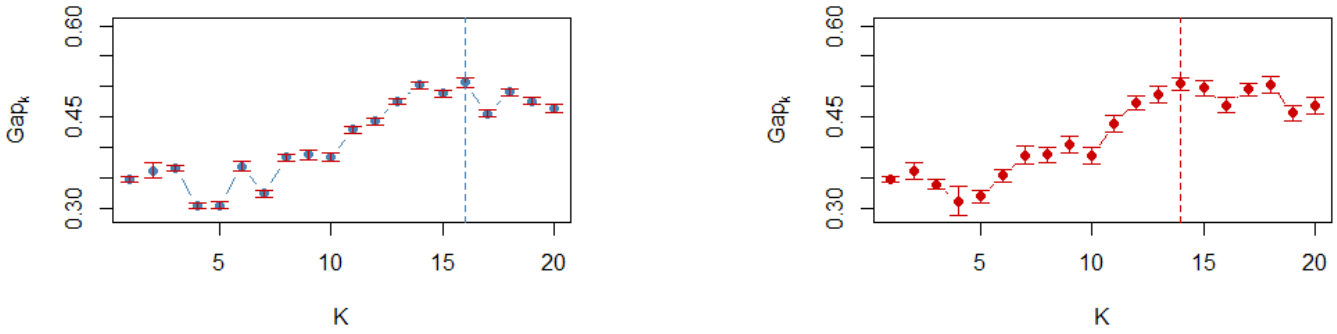
Figure 6: Distribution of the optimal K across 100 initializations

Figures 5 and 6 illustrate how the optimal number of clusters selected by the K-means and K-medoids algorithms varies across different initializations. For K-means, the most frequently chosen value was K = 15, which consistently produced the highest average silhouette score. This indicates that, across both 50 and 100 runs, K-means reliably converged to the same solution, suggesting high stability in this case. This is surprising as the algorithm is typically sensitive to random initial centroids.

In contrast, K-medoids showed similar consistency, with K = 20 emerging as the optimal number of clusters across all runs. This reinforces the known robustness of the algorithm, which uses actual data points as medoids rather than calculated means. Since medoids are less influenced by outliers or initialization, K-medoids naturally tends to be more stable.

Selecting K using Gap

The Gap Statistic compares the within-cluster dispersion of our data to that of a reference null distribution. The optimal (smallest) K is where $\text{Gap}(K) \geq \text{Gap}(K+1) - s_{(K+1)}$ where $s_{(K+1)}$ is the standard error.



(a) K-means

(b) K-medoids

Figure 7: Gap statistic for K-means and K-medoids

Compared to the average silhouette plots in Figure 4, Figure 7a shows more variability, making it harder to identify a clear optimal number of clusters for K-means. However, the highest Gap statistic for K-means occurs at $K = 16$, suggesting this as a strong candidate for the optimal number of clusters. Beyond $K = 16$, the Gap values and their associated error bars begin to stabilise. This suggests that adding more clusters offers diminishing returns and may risk overfitting.

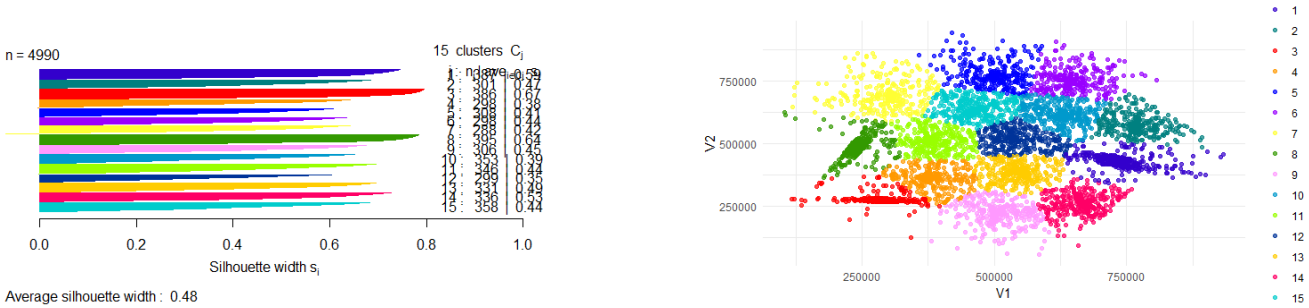
In Figure 7b, we observe that the Gap statistic generally increases up to $K = 14$, after which it begins to plateau and decrease slightly. This pattern is consistent with the average silhouette trend seen in Figures 3b. The increasing Gap statistic reflects a reduction in within-cluster dispersion (WCSS), which is expected as more clusters typically provide a tighter fit to the data.

This suggests that for both algorithms, a cluster range between K=14 and K=16 balances model fit and generalisability.

Cluster Analysis

Silhouette Score Analysis

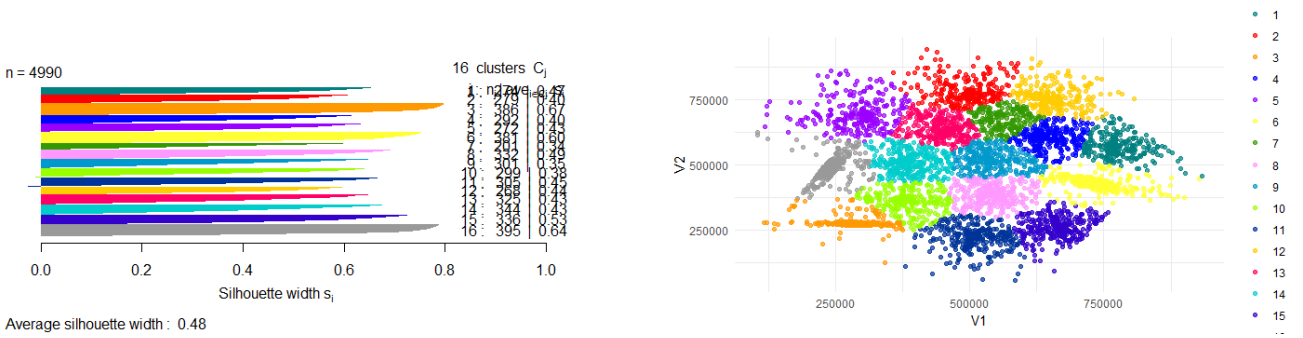
To analyse and compare the cluster qualities, we used our hyper-parameter tuning results to select 2 configurations for K-means ($K = 15$ and $K = 16$) and 2 for K-medoids ($K = 20$ and $K = 14$). We selected $K = 15$ for K-means and $K = 20$ for K-medoids clustering because those were the optimal cluster numbers when increasing initialisation for the respective methods. We also chose $K = 16$ for K-means and $K = 14$ for K-medoids clustering because those were the optimal number of clusters when using Gap statistics.



(a) Silhouette plot with 15 clusters

(b) 15 clusters using K-means algorithms

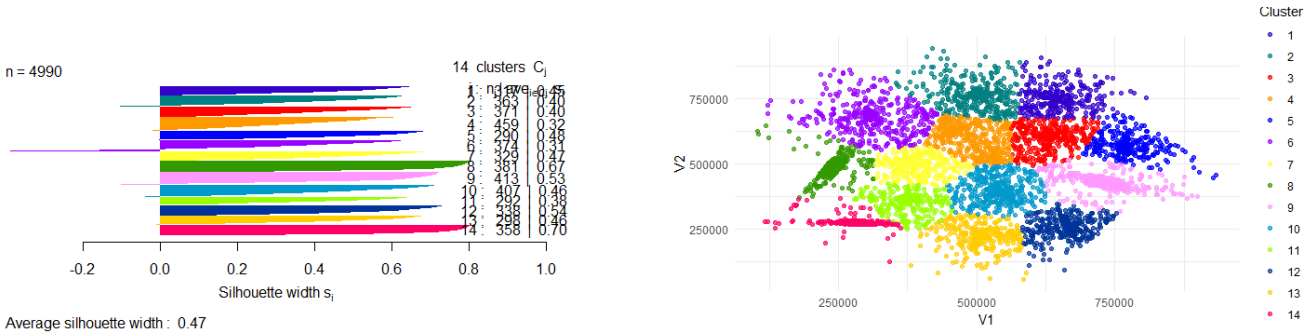
Figure 8: Silhouette Score Analysis for K-means algorithm using 15 clusters



(a) Silhouette plot with 16 clusters

(b) 16 clusters using K-means algorithms

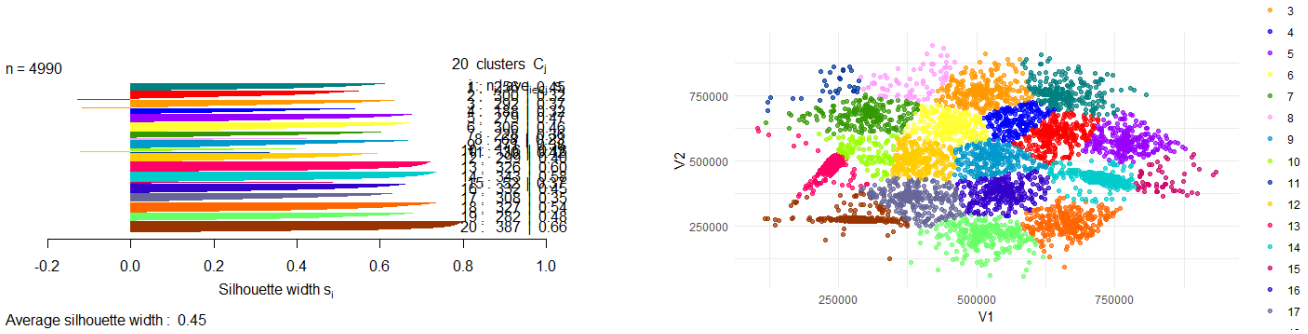
Figure 9: Silhouette Score Analysis for K-means algorithm using 16 clusters



(a) Silhouette plot with 14 clusters

(b) 14 clusters using K-medoids algorithms

Figure 10: Silhouette Score Analysis for K-medoids algorithm using 14 clusters



(a) Silhouette plot with 20 clusters

(b) 20 clusters using K-medoids algorithms

Figure 11: Silhouette Score Analysis for K-medoids algorithm using 20 clusters

From Figure 8, we can see that the highest average silhouette score of 0.48 is produced for the K-means algorithm when using 15 clusters. This score implies weak to moderate cluster separation and suggests that although clusters are distinguishable, they may have noise or overlapping boundaries, which we can also visually observe in Figure 8b.

Similarly, Figure 9, representing the K-means algorithm for 16 clusters, produces the same average silhouette score of 0.48. This implies no meaningful gain from increasing K to 16 from 15, and that the algorithm is likely splitting natural clusters into artificial groups.

Figure 10 displays the silhouette and cluster plot for K-medoids using 14 clusters. This method produces an average silhouette score of 0.47, which is slightly worse than the results produced by the K-means algorithm for both $k = 15$

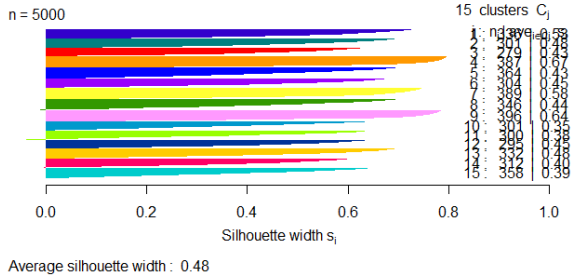
and $K = 16$. Since K-medoids is more robust to outliers, its lower scores may indicate that the true clusters in the data are not well separated.

Lastly, Figure 11 represents the K-medoids clustering algorithm for 20 clusters, however, it produces the worst average silhouette score of 0.45. Since its score is lower than when clustering for $K = 14$, it implies that there is overfitting and that small clusters may represent noise rather than true clusters.

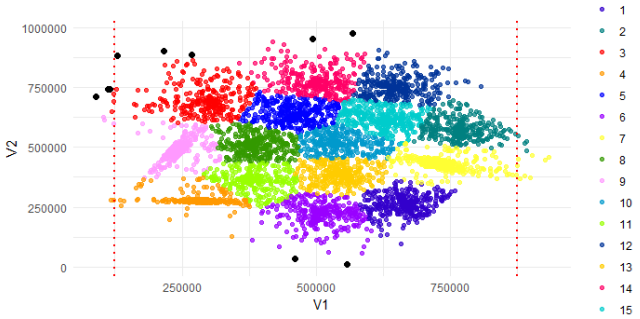
It can also be observed from the Silhouette plots that the K-medoids clustering algorithm contains more clusters consisting of observations with negative scores compared to the K-means clustering algorithm. From comparing Figure 10a and 11a, the K-medoids algorithm produces more negative scores when 14 clusters are constructed compared to 20 clusters, even though K-medoids with 14 clusters produces a higher average silhouette score. This suggests that the trade-off between cohesion and separation of clusters is better balanced when K-medoids has 14 clusters compared to 20.

Therefore, overall, the best cluster assignment from our results is the K-means algorithm using 15 clusters, because it is the simplest structure that produces the highest average silhouette score.

Outlier Analysis



(a) Silhouette plot with 15 clusters (including outliers)



(b) 15 clusters using K-means algorithm (including identified outliers as black cirlces) and potential outlier regions displayed by the vertical red dotted lines

Figure 12: Silhouette Score Analysis for K-means algorithm using 15 clusters and including the identified and potential outliers

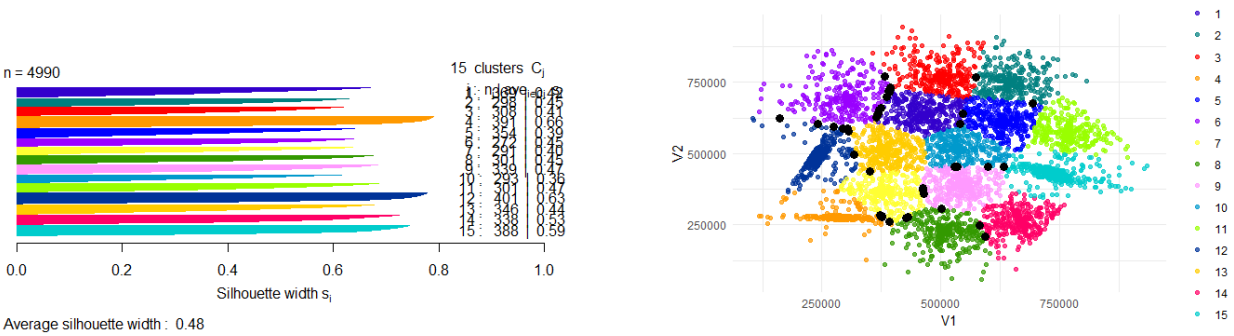
Figure 12b locates the 10 previously identified outliers within our best cluster assignment, and Figure 12a indicates that the average silhouette score has remained the same at 0.48 when the outliers are included. Figure 15, in the appendix, also emphasises that the centroid positioning of K-means clustering when $K = 15$ is not influenced greatly by the outliers., Therefore, our identified outliers are not as influential as we believed them to be.

Possible reasons for this could be that since we use `nstart` of 100, it reduced the chance of outliers disproportionately affecting the final clusters, and since out outliers were near the edges of existing clusters (3, 14, and 6), they did not pull the centroids significantly (as seen in Figure 15, in the Appendix).

Other potential outliers could be observations with $V_1 < 125000$ or $V_1 > 875000$, as depicted by the vertical red dotted lines in Figure 12b.

Post-Processing

The observations with negative scores were identified and reassigned to the cluster with the closest centroid (using euclidean distance), provided that the closest cluster is different to the observations' original cluster.



(a) Silhouette k=15 with cluster reassignment

(b) Cluster k=15 with cluster reassignment

Figure 13: Cluster reassignment of negative silhouette scores for K-means algorithm when $K = 15$

Figure 13b displays the observations that underwent reassignment, and Table 4, in the Appendix, indicates all reassignments of all the negative observations, and it displays their new cluster assignments and updated scores. From those results, we can see that most of the reassignments have improved as their scores are now positive, whilst some observation scores remained negative. Despite this improvement, Figure 13a suggests that the average silhouette score remained the same.

Conclusion

Through this report, we have conducted a comprehensive cluster analysis on a dataset to evaluate the performance of 2 clustering algorithms (K-means and K-medoids). We first performed an exploratory data analysis on the data to prepare them by selecting to use Euclidean distance as our distance metric, analyzing the shape of the univariate and bivariate distributions, identifying and removing outliers, and testing whether further adjustments were required for correlation analysis and scaling of our data, however they were not needed.

Through hyperparameter testing, we investigated average silhouette plots, initialisation sensitivity, increasing initialisations and selecting K using Gap. The results of this investigation suggested that the optimal number of clusters for K-means clustering was 15 or 16, and the optimal number for K-medoids was 14 or 20.

Thereafter, we conducted our cluster analysis and found that the K-means algorithm performed better than the K-medoids algorithm as it produced higher average silhouette scores (0.48) for both $K = 15$ and $K = 16$. Therefore, for the rest of our cluster analysis, we chose our best cluster assignment to be the K-means algorithm using $K = 15$ as our optimal number of clusters. Usually, K-means clustering is susceptible to outliers; however, after further investigation, we found that our initially identified 10 outliers had minimal influence on our cluster analysis. This does not imply that K-means treated our outliers correctly, but rather that our chosen outliers were not very influential.

On the contrary, K-medoids demonstrated robustness to noise and outliers by using the actual data observations as centers, thus allowing it to maintain stable cluster boundaries. However, in regions with overlapping points or where 2 clusters blend, K-medoids might have anchored points that do not improve separation, which therefore could have led to lower cohesion.

If time allowed, a way to improve workflow is to correctly identify outliers. We could use Isolation Forest or DBSCAN to pre-filter extreme points before performing the cluster analysis, and this should improve our results for the K-means algorithm. Alternatively, we could apply a hybrid approach, where we would use K-means clustering to find the initial centroids, then use those centroids as starting points for the K-medoids clustering algorithm. This approach could improve our results as it would combine the strengths of both algorithms while reducing the impact of their weaknesses.

Appendix

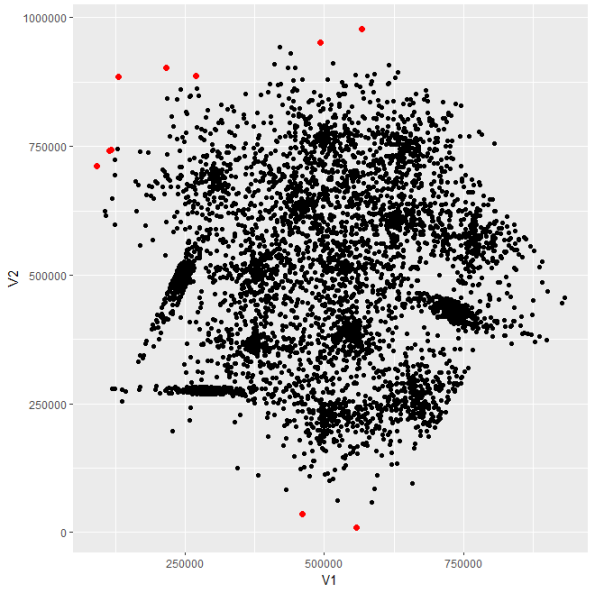


Figure 14: Scatterplot of V_1 and V_2, with the identified outliers in red.

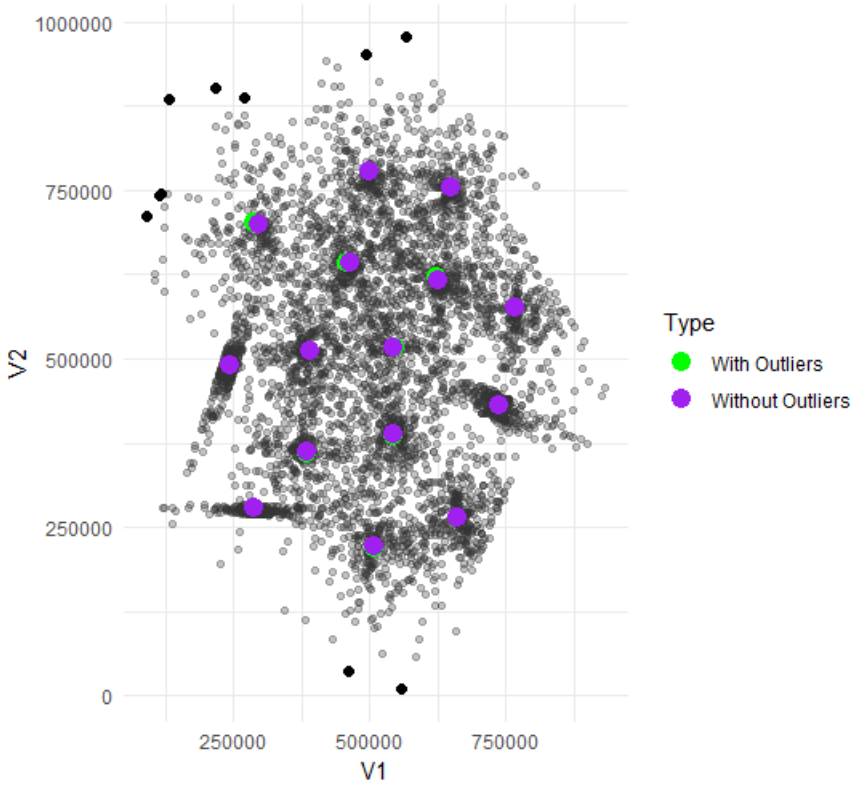


Figure 15: Scatterplot emphasising the difference in centroids of k-mean clustering with outliers (green) and without outliers (purple). The outliers are represented by black circles.

Table 4: Analysis of observations with negative scores

Index	Original Score	Original Cluster	Re-assigned Cluster	New Score
223	−0.0054	14	10	0.0066
277	−0.0137	11	14	0.0195
338	−0.0212	8	13	0.0397
442	−0.0570	8	13	0.0789
492	−0.0238	8	7	−0.0139
507	−0.0370	8	13	0.0612
533	−0.0029	12	7	0.0066
552	−0.0056	8	7	−0.0060
563	−0.0487	8	13	0.0694
1325	−0.0098	9	13	−0.0161
1377	−0.0178	9	3	0.0196
1525	−0.0095	15	2	0.0094
1655	−0.0022	4	1	−0.0010
1849	−0.0053	4	1	0.0038
1955	−0.0228	15	6	0.0321
1960	−0.0051	15	12	−0.0071
2035	−0.0336	15	6	0.0445
2123	−0.0193	15	2	0.0188
2198	−0.0208	15	6	0.0330
2311	−0.0051	4	15	0.0023
2507	−0.0185	4	15	0.0151
2651	−0.0583	8	13	0.0813
2808	−0.0125	12	7	0.0063
2840	−0.0431	8	13	0.0660
3031	−0.0540	15	6	0.0658
3076	−0.0431	15	6	0.0549
3313	−0.0480	8	7	0.0460
3355	−0.0566	8	13	0.0802
3361	−0.0340	8	13	0.0540
3438	−0.0131	8	13	0.0347
3931	−0.0008	10	13	−0.0240
3989	−0.0036	9	2	0.0065
3990	−0.0198	9	2	0.0224
4073	−0.0274	4	2	0.0221
4106	−0.0088	9	2	0.0052
4158	−0.0096	15	2	0.0087
4238	−0.0177	9	2	0.0206
4380	−0.0701	8	7	0.0754
4425	−0.0668	8	7	0.0757
4804	−0.0163	8	11	0.0157
4810	−0.0690	8	13	0.0900

```
columns
colum1      # Assignment 3
colum2
colum3 library(tidyverse)
colum4 library(ggplot2)
colum5 library(ggExtra)
colum6 library(dplyr)
colum7 library(GGally)
colum8 library(parallel)
colum9 library(cluster)
colum10 library(factoextra)
```

```r

################################################
############ Exploratory Data Analysis ##############

#------------(a) Data Description_________________#

data <- read.table("STA4026_Assignment_Clustering.txt", header = FALSE)

colnames(data) <- c("V1", "V2") # we can decide of the variables later

og_size <- dim(data) # 2 variables with 5000 observations

data_types <-  sapply(data, class) # data type - both integers

missing_vals <-  colSums(is.na(data)) # 0 missing values in both columns

duplicates <- any(duplicated(data)) # false - no duplicates

infinite_vals <- any(sapply(data, function(x) any(is.infinite(x)))) # false - no infinite values

data_summary <- summary(data)

#----------(c) Exploratory Analysis---------------#

pairPlot <- ggplot(data, aes(x=V1, y=V2)) +
  geom_point(alpha = 0.5) +
  labs(x = expression(V[1]), y = expression(V[2])) +
  theme_minimal()
ggMarginal(pairPlot, type = "densigram")


#-----(d) Exploratory Analysis of Distance--------#

dist <- dist(data, method = "euclidean")
dist_vals <- as.vector(dist)

ggplot(data.frame(Distance = dist_vals), aes(x = Distance)) +
  geom_histogram(aes(y = after_stat(count)),
    bins = 30, fill = "lightskyblue", color = "black") +
  geom_density(aes(y = after_stat(count) * (max(dist_vals) - min(dist_vals))/30),
    alpha = 0.2, color = "red") +
  labs(x = "Distance",
       y = "Frequency") +
  theme_minimal()

#----------(e) Outlier Identification-------------#

data_clean <- data |>
  mutate(dist_from_median = sqrt((V1 - median(V1))^2 + (V2 - median(V2))^2)) |>  mutate(
    observation = row_number())

outliers <-  data_clean |>
  arrange(desc(dist_from_median)) |>
  head(10)

data_cleaned <- anti_join(data, outliers)

ggplot(data, aes(V1, V2)) +
  geom_point() +
  geom_point(data = outliers, color = "red", size = 2)

#----------(f) Correlation Analysis---------------#
cor_mat <- cor(data_cleaned$V1, data_cleaned$V2)


################################################
############ Hyper-parameter Tuning ###############

```

```r
#--------------(a) Selecting K--------------------#

# Range of K values
K <- 2:20

# compute distance matrix once
dists <- dist(data_cleaned)

### ---- K-MEANS ---- ###
sil_widths_kmeans <- numeric(length(K))

for (i in seq_along(K)) {
  k <- K[i]
  km <- kmeans(data_cleaned, centers = k, nstart =100)
  sil <- silhouette(km$cluster, dists)
  sil_widths_kmeans[i] <- mean(sil[, 3])
}

# Find the best K (max silhouette)
best_k_kmeans <- K[which.max(sil_widths_kmeans)]

# Plot for K-means
plot(K, sil_widths_kmeans, type = 'b', pch = 19, col = 'blue',
     xlab = "Number of Clusters K", ylab = "Average Silhouette Score",
     main = "")
abline(v = best_k_kmeans, lty = 2, col = "darkblue")

### ---- K-MEDOIDS (CLARA) ---- ###
sil_widths_medoid <- numeric(length(K))

for (i in seq_along(K)) {
  k <- K[i]
  km <- clara(data_cleaned, k = k, metric = "euclidean", pamLike = TRUE, samples = 50)
  sil <- silhouette(km$clustering, dists)
  sil_widths_medoid[i] <- mean(sil[, 3])
}

# Best K for medoids
best_k_medoids <- K[which.max(sil_widths_medoid)]

# Plot for K-medoids
plot(K, sil_widths_medoid, type = 'b', pch = 19, col = 'red',
     xlab = "Number of Clusters K", ylab = "Average Silhouette Score",
     main = "")
abline(v = best_k_medoids, lty = 2, col = "darkred")

#----------(b) Initialisation Sensitivity---------#

set.seed(2025)  # reproducibility

iter <- 50 # number of iterations

# Define K-means optimal K function
opt_kmeans <- function(data, k_range) {
  sil_width <- sapply(k_range, function(k) {
    km <- kmeans(data, centers = k, nstart = 50, iter.max = 50)
    sil <- silhouette(km$cluster, dist(data))
    mean(sil[, "sil_width"])
  })
  k_range[which.max(sil_width)]
}

# Define K-medoids optimal K function
opt_kmed <- function(data, k_range) {
  sil_width <- sapply(k_range, function(k) {
    cl <- clara(data, k, metric = "euclidean", samples = 50, pamLike = TRUE)
    cl$silinfo$avg.width
  })
```

```
  k_range[which.max(sil_width)]
}

# Create cluster for parallel processing
cl <- makeCluster(detectCores() - 1)

# Export variables and functions to the cluster
clusterExport(cl, c("opt_kmeans", "opt_kmed", "data_cleaned", "k_vals", "silhouette", "dist", "kmeans", "clara"))

# Parallel runs for K-means
opt_kmeans_pll <- parSapply(cl, 1:iter, function(i) opt_kmeans(data_cleaned, K))

# Parallel runs for K-medoids
opt_kmed_pll <- parSapply(cl, 1:iter, function(i) opt_kmed(data_cleaned, K))

stopCluster(cl)  # Stop cluster

# Frequency tables
kmeans_freq <- table(opt_kmeans_pll)
kmedoids_freq <- table(opt_kmed_pll)

# Plot K-means results #Optimal Number of Clusters (K*) - K-means (Parallel 100 runs)
HPT_kmeans <- ggplot(data.frame(K = names(kmeans_freq), Frequency = as.integer(kmeans_freq)), aes(x = K, y = Frequency)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  labs(title = "",
       x = "Optimal number of clusters (K*)",
       y = "Frequency") +
  theme_minimal() +
  theme(panel.border = element_rect(colour = "black", fill = NA, size = 0.7),
        axis.line = element_line(color = "black"),
        axis.ticks = element_line(color = "black"),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())

# Plot K-medoids results Optimal Number of Clusters (K*) - K-medoids (Parallel 100 runs)
HPT_kmeds <- ggplot(data.frame(K = names(kmedoids_freq), Frequency = as.integer(kmedoids_freq)), aes(x = K, y = Frequency)) +
  geom_bar(stat = "identity", fill = "#CC0000") +
  labs(title = "",
       x = "Optimal number of clusters (K*)",
       y = "Frequency") +
  theme_minimal() +
  theme(panel.border = element_rect(colour = "black", fill = NA, size = 0.7),
        axis.line = element_line(color = "black"),
        axis.ticks = element_line(color = "black"),
        panel.grid.major = element_blank(),
        panel.grid.minor = element_blank())

# Save plots (optional)
ggsave("HPT_kmeans.png", HPT_kmeans)
ggsave("HPT_kmeds.png", HPT_kmeds)

#---------(c) Increasing Initialisations----------#

cl <- makeCluster(detectCores() - 1)
clusterExport(cl, c("opt_kmeans", "data_cleaned", "k_vals", "silhouette", "dist", "kmeans"))
opt_kmeans_pll <- parSapply(cl, 1:200, function(i) opt_kmeans(data_cleaned, K))
stopCluster(cl)

parallel_k_freq <- table(opt_kmeans_pll)

#Distribution of Optimal K from 200 Parallel Initialisations
Kmean_200<-ggplot(data.frame(K = names(parallel_k_freq), Frequency = as.integer(parallel_k_freq)), aes(x = K, y = Frequency)) +
  geom_bar(stat = "identity", fill = "darkgreen") +
  labs(title = "",
```

```
       x = "Optimal number of clusters (K*)",
       y = "Frequency") +
  theme_minimal()

# Create cluster with number of cores minus one
cl <- makeCluster(detectCores() - 1)
clusterExport(cl, c("opt_kmed", "data_cleaned", "k_vals", "clara"))
opt_kmed_pll <- parSapply(cl, 1:200, function(i) {
  opt_kmed(data_cleaned, K)
})

# Stop the cluster after finishing
stopCluster(cl)

# Distribution of Optimal K from 300 Parallel Initialisations (K-medoids)
opt_kmed_pll_freq <- table(opt_kmed_pll)

Kmed_200<-ggplot(data.frame(K = names(opt_kmed_pll_freq), Frequency = as.integer(opt_kmed_pll_freq)), aes(x = K, y = Frequency)) +
  geom_bar(stat = "identity", fill = "#CC0000") +
  labs(title = "",
       x = "Optimal number of clusters (K*)",
       y = "Frequency") +
  theme_minimal()

ggsave("HPT_kmeans_300.png", Kmean_300)
ggsave("HPT_kmeds_300.png", Kmed_300)

#----------(d) Selecting K using Gap--------------#
# Computing the gap statistic
set.seed(2025)
km_gap_stat <- clusGap(data_cleaned, FUN = kmeans, K.max = 20, B = 50)

kmed_gap_stat <- clusGap(data_cleaned, FUN = clara, K.max = 20, B = 50)
# Find the index of max gap value
best_k_kmeans <- which.max(km_gap_stat$Tab[,"gap"])
best_k_medoids <- which.max(kmed_gap_stat$Tab[,"gap"])


km_gap<-plot(km_gap_stat, main="", xlab="K", col="steelblue", pch=16, ylim = c(0.29,0.6))
kmed_gap<-plot(kmed_gap_stat, main="", xlab="K", col="#CC0000", pch=16, ylim = c(0.29,0.6))


plot(km_gap_stat, main="", xlab="K", col="steelblue", pch=16, ylim=c(0.29,0.6))
abline(v = best_k_kmeans, lty=2, col="steelblue")

plot(kmed_gap_stat, main="", xlab="K", col="#CC0000", pch=16, ylim=c(0.29,0.6))
abline(v = best_k_medoids, lty=2, col="#CC0000")

ggsave("km_gap.jpeg", km_gap)
ggsave("kmed_gap.jpeg", kmed_gap)

best_k_kmeans <- km_gap_stat$Tab[which.max(km_gap_stat$Tab[,"gap"]), "k"]
best_k_medoids <- kmed_gap_stat$Tab[which.max(kmed_gap_stat$Tab[,"gap"]), "k"]

#####################################################
################Cluster Analysis###################
#---------(a) Silhouette Score Analysis-----------#

#.................. K MEANS 15.....................#

set.seed(2025)
colours15 <- c("#3300CC","#008080","#FF0000", "#FF9900","#0000FF", "#9900FF",
               "#FFFF33", "#339900", "#FF99FF", "#0099CC", "#99FF00",
               "#003399","#FFCC00", "#FF0066","#00CCCC")

km15 <- kmeans(data_cleaned, centers = 15, nstart = 100, iter.max = 50) # K-means with 15 cluster
```

```r

data_cleaned$Cluster <- as.factor(km15$cluster)

# Compute silhouette widths
sil15 <- silhouette(km15$cluster, dist(data_cleaned))
data_cleaned$SilWidth = sil15[, 'sil_width']

plot(sil15, col = colours15, border = NA, main = "")

data_cleaned$Cluster <- as.factor(km15$cluster)

# Scatter plot of the clustered data
cluster_plot <- ggplot(data_cleaned, aes(x = V1, y = V2, color = Cluster)) +
  geom_point(alpha = 0.7) +
  theme_minimal() +
  labs(title = "",
       x = "V1",
       y = "V2") +
  scale_color_manual(values = colours15) # Applying custom colors

# Print the plot
print(cluster_plot)

#.............. K MEANS 16.....................#

set.seed(2025)
colours16 <- c("#008080","#FF0000", "#FF9900","#0000FF", "#9900FF",
                "#FFFF33", "#339900", "#FF99FF", "#0099CC", "#99FF00",
                "#003399","#FFCC00", "#FF0066","#00CCCC","#3300CC","#999999")

km16 <- kmeans(data_cleaned, centers = 16, nstart = 100, iter.max = 50) # K-means with 16
column    cluster

data_cleaned$Cluster16 <- as.factor(km16$cluster)

# Compute silhouette widths
sil16 <- silhouette(km16$cluster, dist(data_cleaned))

plot(sil16, col = colours16, border = NA, main = "")

data_cleaned$Cluster16 <- as.factor(km16$cluster)

# Scatter plot of the clustered data
cluster_plot <- ggplot(data_cleaned, aes(x = V1, y = V2, color = Cluster16)) +
  geom_point(alpha = 0.7) +
  theme_minimal() +
  labs(title = "",
       x = "V1",
       y = "V2") +
  scale_color_manual(values = colours16) # Applying custom colors

# Print the plot
print(cluster_plot)

#.............K MEDOIDS 14.....................#

set.seed(2025)
colours14 <- c("#3300CC","#008080","#FF0000", "#FF9900","#0000FF", "#9900FF",
               "#FFFF33", "#339900", "#FF99FF", "#0099CC", "#99FF00",
               "#003399","#FFCC00", "#FF0066")

cl14 <- clara(data_cleaned, 14, metric= "euclidean", samples = 50, pamLike=T)

sil14c <- silhouette(cl14$clustering, dist(data_cleaned))

plot(sil14c, col = colours14, border = NA, main = "")

data_cleaned$Cluster14c <- as.factor(cl14$clustering)
```

```r

# Scatter plot of the clustered data_cleaned
cluster_plot <- ggplot(data_cleaned, aes(x = V1, y = V2, color = Cluster14c)) +
  geom_point(alpha = 0.7) +
  theme_minimal() +
  labs(title = "",
       x = "V1",
       y = "V2",
       color="Cluster") +
  scale_color_manual(values = colours14) # Applying custom colors

# Print the plot
print(cluster_plot)

#.................K MEDOIDS 20.....................#

set.seed(2025)
colours20 <- c(
  "#008080", "#FF0000", "#FF9900", "#0000FF", "#9900FF",
  "#FFFF33", "#339900", "#FF99FF", "#0099CC", "#99FF00",
  "#003399", "#FFCC00", "#FF0066", "#00CCCC", "#CC0066", "#3300CC",
  "#666699", "#FF6600", "#66FF66", "#993300"
)


cl20 <- clara(data_cleaned, 20, metric= "euclidean", samples = 50, pamLike=T)

sil20c <- silhouette(cl20$clustering, dist(data_cleaned))

plot(sil20c, col = colours20, border = NA, main = "")

data_cleaned$Cluster20c <- as.factor(cl20$clustering)

# Scatter plot of the clustered data
cluster_plot <- ggplot(data_cleaned, aes(x = V1, y = V2, color = Cluster20c)) +
  geom_point(alpha = 0.7) +
  theme_minimal() +
  labs(title = "",
       x = "V1",
       y = "V2",
       color="Cluster") +
  scale_color_manual(values = colours20) # Applying custom colors

# Print the plot
print(cluster_plot)

#--------------(b) Outlier analysis---------------#
# locate outliers on cluster and silhouette plots
set.seed(2025)

km_with_outliers <- kmeans(data, centers = 15, nstart = 100, iter.max = 50) # K-means with 15
column    cluster on data with outliers

outlier_indices <- c(461, 340, 2540, 4722, 442, 486, 501, 2317, 535, 4723)

data$Cluster <- as.factor(km_with_outliers$cluster)

# Compute silhouette widths
sil_with_outliers <- silhouette(km_with_outliers$cluster, dist(data))
data$SilWidth = sil_with_outliers[, 'sil_width']

plot(sil_with_outliers, col = colours15, border = NA, main = "")

# Scatter plot of the clustered data
cluster_plot <- ggplot(data, aes(x = V1, y = V2, color = Cluster)) +
  geom_point(alpha = 0.7) +
  geom_vline(xintercept = c(125000, 875000),
             color = "red",
```

```
            linetype = "dotted",
            linewidth = 0.8) +
  theme_minimal() +
  labs(title = "",
       x = "V1",
       y = "V2") +
  scale_color_manual(values = colours15) +
  geom_point(data[outlier_indices, ],
            mapping = aes(x = V1, y=V2, col = "black"),
            col ="black", fill = "black", size = 2)

# Print the plot
print(cluster_plot)

##### compare original cluster with outliers to clustering without outliers
common_cols <- intersect(colnames(data), colnames(data_cleaned))
data_subset <- data[, common_cols]
data_cleaned_subset <- data_cleaned[, common_cols]

# original cluster
set.seed(2025)
km_with_outliers <- kmeans(data_subset, centers = 15, nstart = 100, iter.max = 50)

# cluster without outliers
km_wo_outliers <- kmeans(data_cleaned_subset, centers = 15, nstart = 100, iter.max = 50)

# compare cluster centers
all_centers <- rbind(
  data.frame(km_with_outliers$centers, Type = "With Outliers"),
  data.frame(km_wo_outliers$centers, Type = "Without Outliers")
)

ggplot() +
  geom_point(data = data, aes(x = V1, y = V2), color = "gray20", alpha = 0.3) +
  geom_point(data = all_centers, aes(x = V1, y = V2, color = Type), size = 4) +
  scale_color_manual(values = c("With Outliers" = "green", "Without Outliers" = "purple")) +
  labs(x = "V1", y = "V2") +
  geom_point(data[outlier_indices, ],
            mapping = aes(x = V1, y=V2, col = "black"),
            col ="black", fill = "black", size = 2) +
  theme_minimal()

#--------------(c) Post-Processing----------------#

km15 <- kmeans(data_cleaned, 15, nstart = 100, iter.max = 50)

# silhouette scores
sil15 <- silhouette(km15$cluster, dist(data_cleaned))
negative_indices <- which(sil15[, 3] < 0) # negative silhouette points

# reassign neg score obs, to closest different cluster (based on centroid diff)
centroids <- km15$centers
updated_clusters <- km15$cluster

for (i in negative_indices) {
  curr_cluster <- km15$cluster[i]
  dist <- sqrt(rowSums((t(t(centroids) - unlist(data_cleaned[i, ]))^2))) # calc dist to all
centroids
  alt_clusters <- order(dist) # get dist to other clusters (asc - closest to furthest)
  alt_clusters <- alt_clusters[alt_clusters != curr_cluster] # ignore distance to its current
cluster

  # only reassing if a better cluster exists
  if (length(alt_clusters) > 0) {
    updated_clusters[i] <- alt_clusters[1] # assign to closest diff cluster
  }
}
```

```r
# creating a new data frame with updated clusters
new_df <- cbind(data_cleaned, cluster = as.factor(updated_clusters))

# plot the clusters with the re-assigned obs in black
ggplot(new_df, aes(x = V1, y = V2, color = cluster)) +
  geom_point(alpha = 0.7) +
  geom_point(data = new_df[negative_indices, ],
             aes(x = V1, y = V2),
             color = "black", size = 3) +
  theme_minimal() +
  scale_color_manual(values = colours15) +
  labs(x = "V1", y = "V2", color = "Clusters (Updated)")

# silhouette comparisons
# original silhouette
original_sil <- sil15
plot(original_sil, col = colours15, border = NA, main = "Original Silhouette")

# updated silhouette after reassignment:
updated_sil <- silhouette(updated_clusters, dist(data_cleaned))
plot(updated_sil, col = colours15, border = NA, main = "")

# summary table of negative-score points
results_table <- data.frame(
  Index = negative_indices,
  Original_Silhouette_Score = round(sil15[negative_indices, 3], 4),
  Original_Cluster = sil15[negative_indices, 1],
  Reassigned_Cluster = sil15[negative_indices, 2],
  New_Silhouette_Score = round(updated_sil[negative_indices, 3], 4)
)
```

Listing 1: (R Code for Clustering Analysis)