Assignment 2: Data Modelling

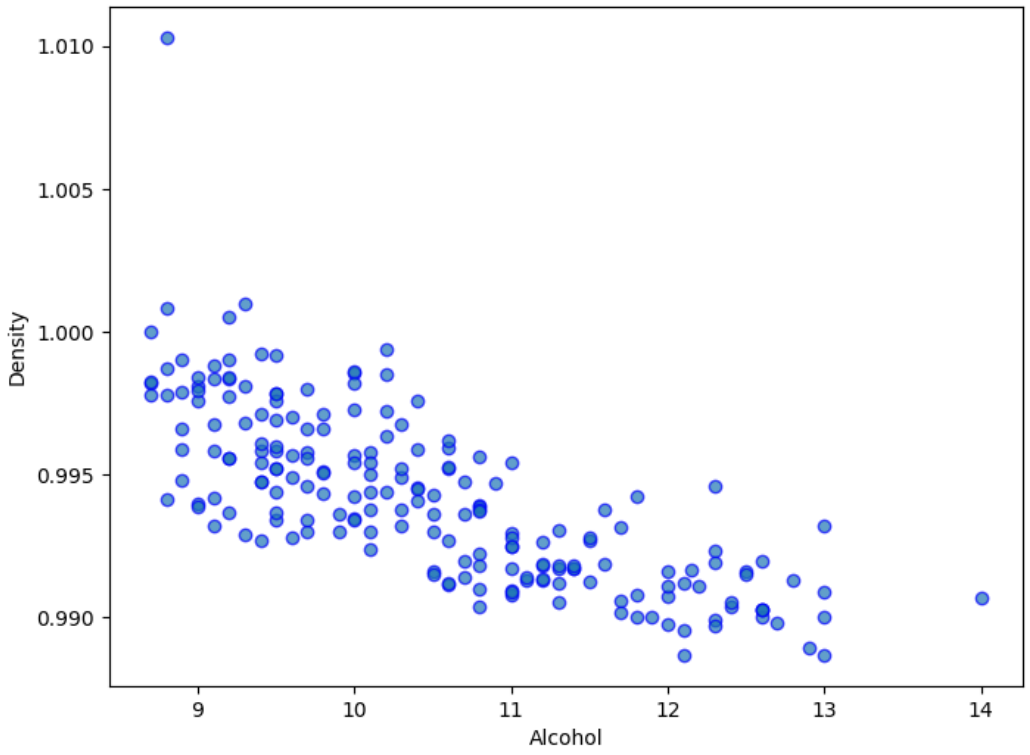
Jung-De Chiou(s4068959)

Task 1: Regression

After extracting the **density** and **alcohol** columns, I cleaned the data to remove any missing values. I then randomly selected 200 samples from the cleaned dataset for analysis. I used a **scatter plot** for the analysis, as it best illustrates the distribution and relationship between the two variables.

Key Observations:

- **Negative Correlation**: There is a clear negative correlation between alcohol and density, meaning that as the alcohol content increases, the density generally decreases.
- **Outliers**: There appears to be an outlier located in the top-left corner of the plot. In this outlier, the alcohol content is lower, and the density is significantly higher compared to the other points. This particular point may represent an unusual sample and may require further investigation.
- **Clustered Distribution**: Most data points are clustered within a fairly narrow range, with alcohol values between 9 and 13 and density values between 0.99 and 1.00. This suggests that the samples within this range share similar properties and are likely characteristic of typical substances in this dataset.
- **Gradual Slope**: The relationship between alcohol and density appears to follow a gradual slope, with no sharp changes or breaks. This indicates a steady and predictable relationship between the two variables.



Simple Linear Model:

We can use **sklearn's** LinearRegression to further fit the regression equation between the two variables, with alcohol being the dependent variable and density as the independent variable. The linear model equation is as follows:

Linear Model Equation: alcohol = 10.4758 + (-0.8950 * density)

Additionally, the model's **R-squared (R^2)** is **0.5738**, and the **Mean Squared Error (MSE)** is **0.5949**.

Key Observations:

- **Negative Slope:** This is consistent with the general trend observed in the scatter plot, where most data points follow this inverse relationship, with alcohol content decreasing as density increases.
- **R-squared (R^2):** Although the model captures the overall trend, the R^2 score of 0.5738 indicates that only about 57.38% of the variability in alcohol content can be explained by the density. This relatively moderate R^2 suggests that the linear model is not fully capturing the relationship between the two variables. This indicates that while density is a significant predictor of alcohol content, other factors may also influence alcohol levels, or there could be some noise in the data affecting the relationship.
- **Mean Squared Error (MSE) Analysis:** The MSE value of 0.5949 represents the average squared difference between the actual and predicted alcohol content values. A smaller MSE would indicate a better fit. However, in this case, the relatively higher MSE suggests that the model's predictions have a moderate amount of error. This further supports the idea that while the linear model is useful in identifying the general trend, it may not be precise enough to make highly accurate predictions, likely due to other unaccounted-for factors or noise in the data.

Task 2: Classification

After cleaning the dataset like in Task 1, I randomly chose 500 samples for analysis. I used all 11 attributes as features, excluding the "quality" attribute as the target variable. I ran an initial KNN model with the default settings, without any adjustments. The training-to-testing ratio was 80:20, and I set k = 5. Below are the results:

kNN

Classification Report:

Quality Scale	Precision	Recall	F1-Score	Support
3	0.00	0.00	0.00	1
4	0.00	0.00	0.00	3
5	0.40	0.68	0.51	28
6	0.48	0.52	0.50	42
7	0.00	0.00	0.00	21
8	0.00	0.00	0.00	5
Accuracy			**0.41**	
Macro Avg	0.15	0.20	0.17	100
Weighted Avg	0.31	0.41	0.35	100

Confusion Metrix

	Predicted 3	Predicted 4	Predicted 5	Predicted 6	Predicted 7	Predicted 8
True 3	0	0	1	0	0	0
True 4	0	0	3	0	0	0
True 5	0	1	19	7	1	0
True 6	0	0	15	22	5	0
True 7	0	0	8	13	0	0
True 8	0	0	1	4	0	0

Accuracy:

- The model achieved an overall accuracy of **41%**, correctly classifying 41 out of 100 test samples. This suggests that the default KNN model performs suboptimally, especially in certain quality categories.

Precision, Recall, and F1-Score:

- **Quality 5 and 6**: These classes have relatively more training samples, so the model's accuracy for these classes is relatively acceptable. However, for the other classes (3, 4, 7, and 8), both Precision and Recall are **0**, which means the model failed to correctly classify any samples for these categories. This suggests that the model has not learned enough information from these categories, likely because of the small number of samples (for example, there is only 1 sample for quality 3 and 3 samples for quality 4), making it difficult for the model to learn effectively.

Confusion Matrix:

- The conclusions drawn from the confusion matrix are similar. Most correctly classified data points are concentrated in the **quality 5 and 6** categories, indicating that the KNN model has some ability to recognize these two categories. However, for **quality 3, 4, 7, and 8**, there were almost no correct classifications. These categories were often misclassified as other categories, particularly quality 5 and 6. Notably, **all 21 samples of quality 7** were misclassified, further indicating that the model struggles to learn effectively for categories with fewer samples.

modified kNN

1. **Parameter Tuning:**

First, I used StandardScaler() to standardize the original sample data. Standardization is important in KNN because it ensures that all features contribute equally to the distance calculation. Since KNN is a distance-based algorithm, features with larger scales (like age or salary) can disproportionately affect the distances and the model's predictions if not standardized.

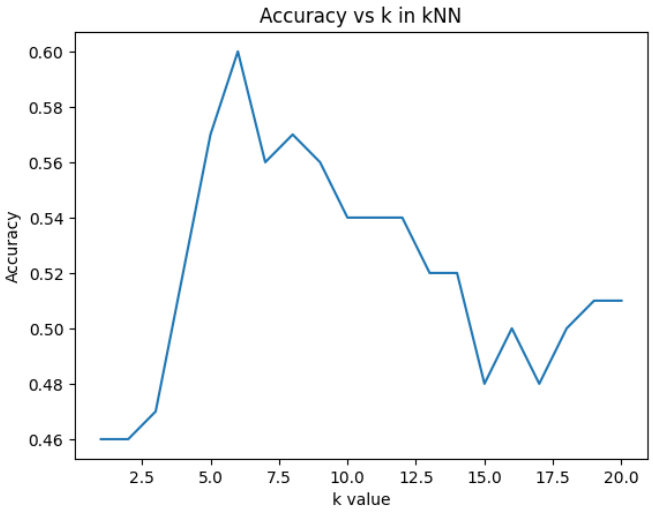
Next, I switched the weight to 'distance' and used p=1 (Manhattan distance). This change means that closer neighbors have more influence on the model's accuracy than farther ones, which is especially helpful when distant neighbors may not be as relevant. In this case, the Manhattan distance, which calculates the absolute difference between points, performed better than Euclidean distance, especially for high-dimensional data or features with varying contributions across dimensions.

As a result of these optimizations, the accuracy increased from 0.41 to 0.57, marking a significant improvement of around 39%.

2. Finding the Optimal k:

To identify the best k value, I looped through different k values and plotted the accuracy scores as a line graph. The result is shown below. We can find that k=6 yields the highest accuracy score of 0.60.

Up to this point, the **accuracy score** has increased from **0.41 to 0.6**. **Quality 5 and 6** remain the two most accurately classified categories. Performance for **quality 7** improved slightly, with an **F1-score** of **0.41**, but there are still many misclassifications. However, **quality 3, 4, and 8** still perform poorly, with no correct classifications for **quality 3 and 8**.



Modified knn Classification Report:

Quality	Precision	Recall	F1-Score	Support
3	0.00	0.00	0.00	1
4	0.00	0.00	0.00	3
5	0.59	0.79	0.68	28
6	0.58	0.76	0.66	42
7	0.75	0.29	0.41	21
8	0.00	0.00	0.00	5
Accuracy				**0.60**
Macro Avg	0.32	0.31	0.29	100
Weighted Avg	0.57	0.60	0.55	100

3. Feature Selection:

I also used the Hill Climbing method for feature selection and reduced the original 11 features to 7. The classification report after feature selection is as follows:

After performing feature selection, we can notice a correct classification for quality 8, which was previously not predicted accurately at all. However, this improvement came with a significant drop in the performance for quality 7, and the overall accuracy also **dropped to 0.43**. The decrease in accuracy and performance after feature selection can be attributed to several factors, such as Hill Climbing' s inability to assess the interactions between features. Some features can contribute to better performance when used together. Therefore, I ran the model with all 11 original features instead of using the selected features. This approach yielded the best results.

Knn Classification Report after feature selection:

Quality	Precision	Recall	F1-Score	Support
3	0.00	0.00	0.00	1
4	0.00	0.00	0.00	3
5	0.41	0.64	0.50	28
6	0.52	0.55	0.53	42
7	0.10	0.05	0.06	21
8	1.00	0.20	0.33	5
Accuracy				**0.43**
Macro Avg	0.34	0.24	0.24	100
Weighted Avg	0.41	0.43	0.39	100

Decision Tree

I applied a **decision tree** model to the same dataset for prediction. I optimized each parameter, including **max_depth**, **min_samples_split**, **min_samples_leaf**, and the criteria (either **gini** or **entropy**). For each parameter, I used a **for loop** to iterate over all reasonable values and output the corresponding **accuracy score**. This helped me determine the best values for each parameter. The final results are as follows:

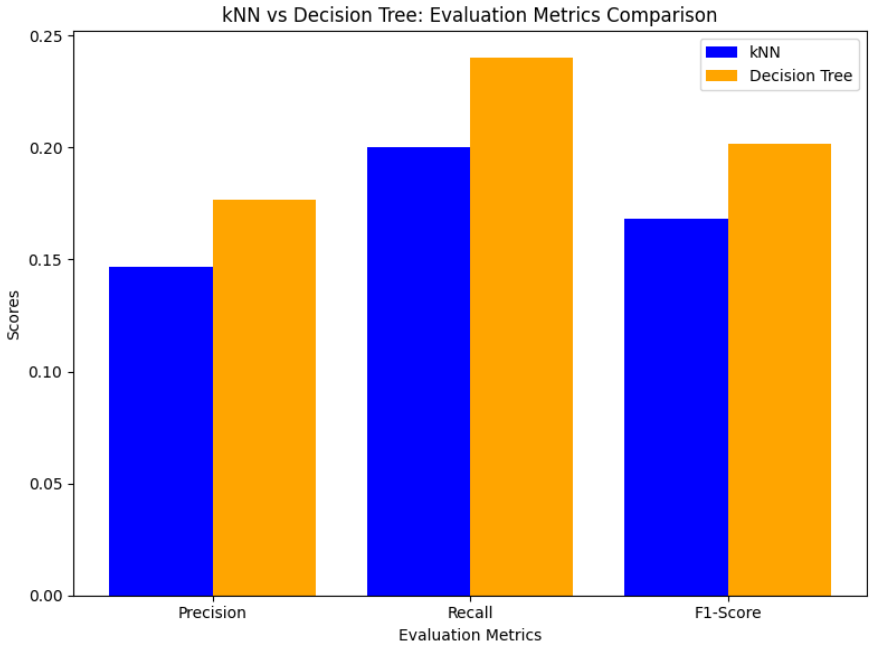
- **Best max_depth**: 3
- **Best min_samples_split**: 6
- **Best min_samples_leaf**: 19
- **Criterion**: 'gini'

DecisionTree Classification Report:

Quality	Precision	Recall	F1-Score	Support
3	0.00	0.00	0.00	1
4	0.00	0.00	0.00	3
5	0.58	0.68	0.62	28
6	0.48	0.76	0.59	42
7	0.00	0.00	0.00	21
8	0.00	0.00	0.00	5
Accuracy				**0.51**
Macro Avg	0.18	0.24	0.20	100
Weighted Avg	0.36	0.51	0.42	100

Comparison

The chart below compares the average precision, recall, and F1 Score of the two models side by side.



The strength of the kNN model is that it is straightforward to implement, making it a good choice for simple classification tasks. It showed good recall performance for the more commonly represented classes, such as quality 5 and 6, indicating its effectiveness in identifying true positive cases in these categories. However, kNN needs help with less-represented classes, like quality 3, 4, 7, and 8, due to its sensitivity to imbalanced data. Additionally, kNN is susceptible to the curse of dimensionality, which means that as the number of features increases, its performance may deteriorate because the distance measures become less meaningful in high-dimensional space.

On the other hand, the Decision Tree model performed better in terms of precision and F1-score, especially for the larger classes, suggesting that it is better at balancing precision and recall to reduce false positives. Decision Trees are also easier to interpret, providing a clear view of the decision-making process and highlighting influential features. However, Decision Trees are prone to overfitting, especially if parameters like tree depth and minimum samples per split are not carefully tuned. While the Decision Tree model can handle imbalanced data slightly better than kNN, it still struggled with the smaller classes in this dataset, failing to provide accurate predictions for categories with fewer samples. Additionally, Decision Trees can be unstable, as small changes in the dataset can lead to different tree structures, which might affect model consistency.

In summary, the Decision Tree model outperforms the kNN model. Decision Tree is better at balancing precision and F1-score, especially for well-represented classes like quality 5 and 6. Unlike kNN, which is affected by the curse of dimensionality and struggles with underrepresented classes, the Decision Tree model can effectively capture relationships between features, allowing it to learn decision boundaries that work well for these larger categories. Additionally, Decision Trees handle complex feature interactions better, offering more interpretability and flexibility. Although

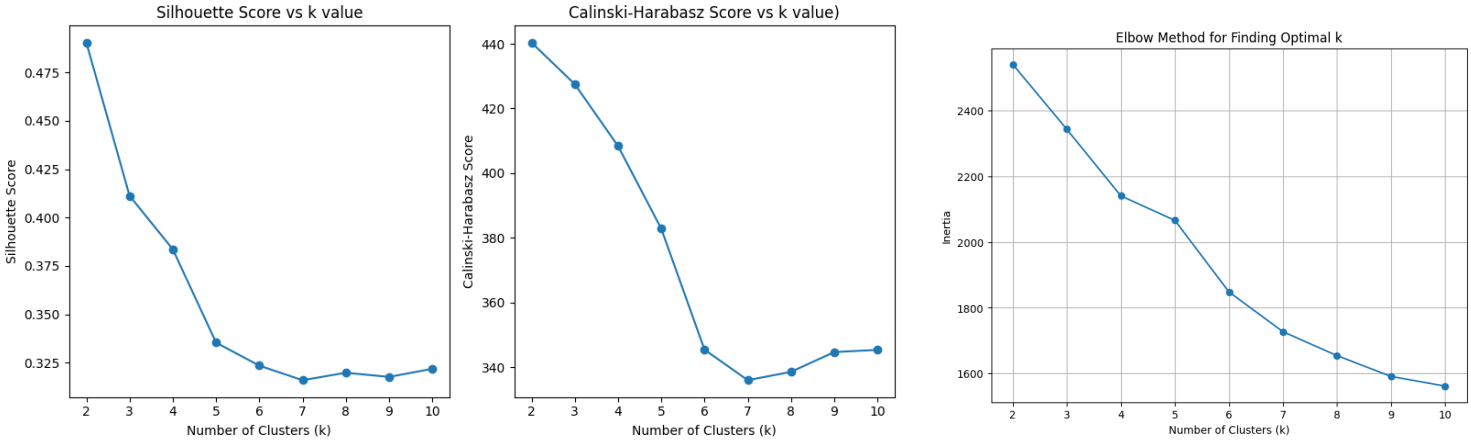
both models struggle with the smaller classes, Decision Tree still manages to deliver more balanced predictions across the dataset, making it the superior choice in this specific context.

Task 3: Clustering

As before, I first cleaned the data, but this time I selected 300 samples to predict using the clustering model.

k-means:

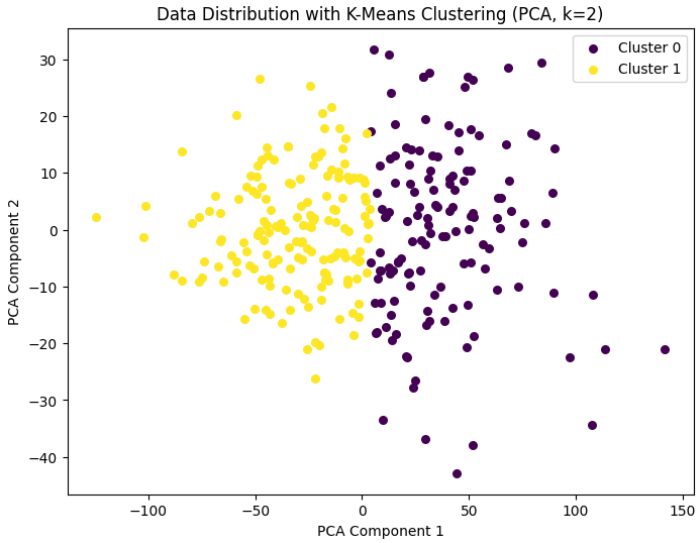
Similarly, I dropped the **'quality'** column, as it is the output variable and can be used for evaluation. Next, I standardized the data, and then the next step was to determine the value of **k**. For this, I used the **Silhouette Score**, **Calinski-Harabasz Index, and Inertia** to evaluate the performance of different **k** values. I tested the performance of **k=2 to 10** using a **for loop** (as shown in the figure below).

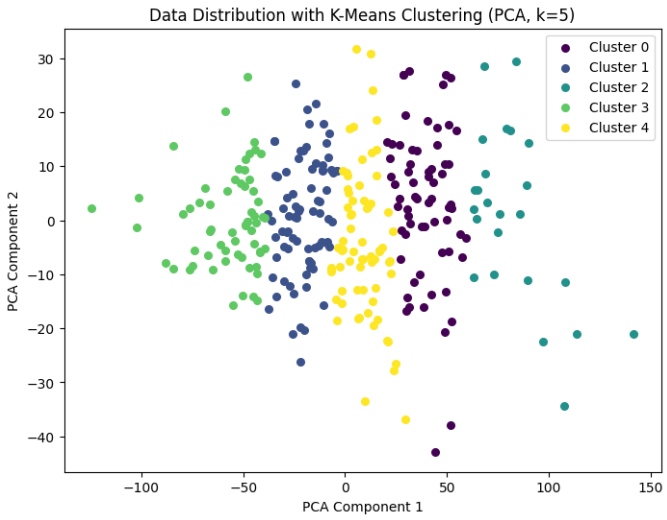


- **Silhouette Score:** This metric measures how similar an object is to its own cluster compared to other clusters. A higher Silhouette Score indicates better-defined clusters. In this case, the Silhouette Score is highest when k=2, suggesting that this value of k results in well-separated and cohesive clusters.
- **Calinski-Harabasz Index:** This index evaluates the ratio of the sum of between-cluster dispersion to within-cluster dispersion. A higher value indicates that clusters are dense and well-separated. Similarly, k=2 yields the highest Calinski-Harabasz score, which further confirms that the two clusters offer the best clustering performance in this scenario.
- Inertia: The inertia metric measures the sum of squared distances between data points and their closest cluster center. Lower inertia values indicate tighter clusters. According to this metric, the optimal value for k is 5. The inertia decreases significantly up to this point, suggesting that four clusters capture more internal variation within the data.

The Silhouette Score and Calinski-Harabasz Index both indicate that k=2 is better for creating more distinct clusters. However, the Inertia suggests that k=4 would better represent the internal structure of the dataset. Therefore, we can try **k=2 or k=5** for the optimal clustering solution.

Afterward, I employed the Principal Component Analysis (PCA) function from sklearn to reduce the 11-dimensional data to 2 dimensions, enabling visualization of the sample data in a scatter plot format. The clusters from k-means were color-coded to display the distribution of each cluster. Below are the results for k=2 and k=5, along with the respective contingency tables:





When k=2, the data is divided into two main clusters, but there is some overlap between them, especially around the center. While most points in each cluster are well-separated, the boundaries between them are not completely distinct. According to the contingency table, cluster 1 contains higher-quality targets (quality 6 and 7), while cluster 0 primarily consists of lower-quality targets (quality 5 and below).

k=2:

Cluster	Target	Count
0	4	3
	5	60
	6	56
	7	15
	8	1
1	3	2
	4	7
	5	43
	6	77
	7	30
	8	6

K=5

Cluster	Target	Count
0	4	2
	5	34
	6	26
	7	4
1	5	22
	6	41
	7	12
	8	4
2	5	12
	6	9
	7	2
	8	1
3	3	2
	4	5
	5	11
	6	28
	7	14
	8	2
4	4	3
	5	24
	6	29
	7	13

When using k=5, the clustering becomes more detailed, spreading the data across five clusters. However, there is still a noticeable overlap, which indicates that some clusters are too close to each other or that the clustering may not fully reflect the natural structure of the data. The contingency table for k=5 shows a better distribution of the data across clusters. Still, it reveals that some clusters, such as clusters 1 and 2, contain mixed target values (quality 5 and 6), suggesting that the clusters are not entirely distinct regarding the output target variable.

Limitations:

- **Overlapping Clusters**: k-Means assumes that clusters are spherical and evenly sized, which can lead to overlap when the data doesn't follow this assumption, especially with irregular cluster shapes or densities.

- **Fixed Number of Clusters**: k-Means requires the number of clusters (**k**) to be predefined, which can be challenging to determine. For example, even though this dataset has 6 known quality categories, setting **k=6** did not result in the best clustering performance, highlighting the difficulty of defining the optimal **k**.
- **Sensitivity to Outliers**: k-Means is easily affected by outliers, which can distort cluster centroids and result in poor clustering outcomes.

Solutions:

- **DBSCAN** can resolve many of these issues by identifying clusters of varying shapes and sizes, automatically detecting the number of clusters based on data density, and treating outliers as noise rather than forcing them into clusters.
- Additionally, increasing the size of the training dataset could help ensure that each **quality** category has sufficient samples, reducing errors and improving overall clustering performance.

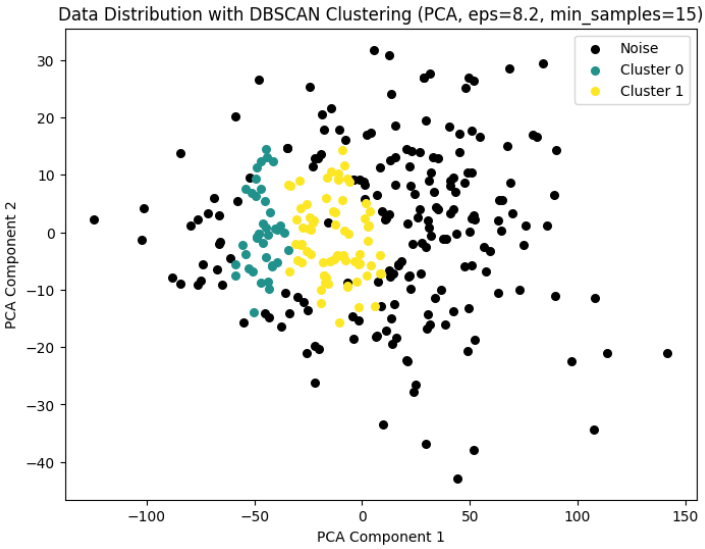
DBSCAN:

I used a method similar to the previous k-means model for DBSCAN. First, I calculated the K-distance for all points using the NearestNeighbors method and then plotted the results to find the elbow point. Based on the graph, it was clear that the best_eps value is around 8.2. Next, I had to determine the best value for min_samples. I accomplished this by looping through min_samples values from 5 to 15 and using three metrics (Silhouette Score, Calinski-Harabasz Index, and Davies-Bouldin Index) to identify which value gives the best performance. The result analysis is as follows:

- **Silhouette Score and Calinski-Harabasz Index:** min_samples=11 achieves the highest Silhouette Score of 0.5003 and the highest Calinski-Harabasz Index of 173.73, indicating that this setting results in the best-defined and well-separated clusters.
- **Davies-Bouldin Index:** A lower Davies-Bouldin Index indicates better clustering. min_samples=6 has the lowest score at 0.445, suggesting better cluster separation in this case. However, min_samples=11 provides a balanced result with a Davies-Bouldin Index of 0.652, making it a strong overall candidate.

Summary:
min_samples=11 yields the best overall performance across all three metrics. It has the highest Silhouette Score and Calinski-Harabasz Index while maintaining a relatively low Davies-Bouldin Index. This suggests that this setting achieves the best balance between well-defined and well-separated clusters.

Next, I used PCA and the plt function to visualize the distribution of clusters after the clustering process. Additionally, I generated a contingency table to evaluate it.



Quality	Noise (-1)	Cluster 0	Cluster 1
3	2	0	0
4	8	1	1
5	77	4	22
6	78	22	33
7	23	10	12
8	2	2	3

- **High Amount of Noise:** A significant portion of the data across all quality levels is classified as noise (-1). This could indicate that The dataset might not be dense enough for DBSCAN to form meaningful clusters for many quality ratings.
- **Clustering for Quality 6 and 7:** DBSCAN performs better for Quality 6 and 7, as it can form meaningful clusters (especially in Cluster 1), although a large number of points are still labeled as noise.
- **Weak Clustering for Lower Qualities (3, 4, and 5):** For quality 3, 4, and 5, DBSCAN struggles to form well-defined clusters, with the majority of points being labeled as noise. This could suggest that these categories are more scattered or lack sufficient density for DBSCAN to detect clusters effectively.

Comparison to K-means:

Quality	K-Means (k=2)	K-Means (k=5)	DBSCAN (Noise)	DBSCAN (Cluster 0)	DBSCAN (Cluster 1)
3	Cluster 1 (2)	Cluster 3 (2)	2	0	0
4	Cluster 0 (3)	Cluster 3 (5)	8	1	1
5	Cluster 0 (60), 1 (43)	Multiple Clusters	77	4	22
6	Cluster 0 (56), 1 (77)	Multiple Clusters	78	22	33
7	Cluster 0 (15), 1 (30)	Multiple Clusters	23	10	12
8	Cluster 0 (1), 1 (6)	Multiple Clusters	2	2	3

Both clustering methods, K-Means and DBSCAN, encountered significant challenges with this dataset. K-Means had difficulty due to its assumption of evenly sized, spherical clusters, resulting in poor separation of certain qualities and mixing within clusters. On the other hand, DBSCAN is sensitive to parameter choices such as eps and min_samples. Despite tuning, it could not handle the full range of qualities in the dataset effectively. While it identified clusters for certain categories, it labeled a large portion of the data as noise, especially for the lower-quality categories. This suggests that the dataset may not be dense enough for DBSCAN's density-based approach.

Result:

Due to its distribution and structure, the dataset may not be suitable for clustering methods like K-Means and DBSCAN. The data does not exhibit clear, distinct clusters required by K-Means, and it lacks sufficient density in certain regions for DBSCAN to be effective. Other approaches, such as supervised learning methods (classification models like Decision Trees or KNN), might yield better results for predicting quality based on the given attributes. Alternatively, more advanced clustering techniques like Gaussian Mixture Models (GMM) or hierarchical clustering could be explored to determine if they offer improved performance.