Assignment 2: Data Modelling

Janna Qian Zi Ng (s4160608)

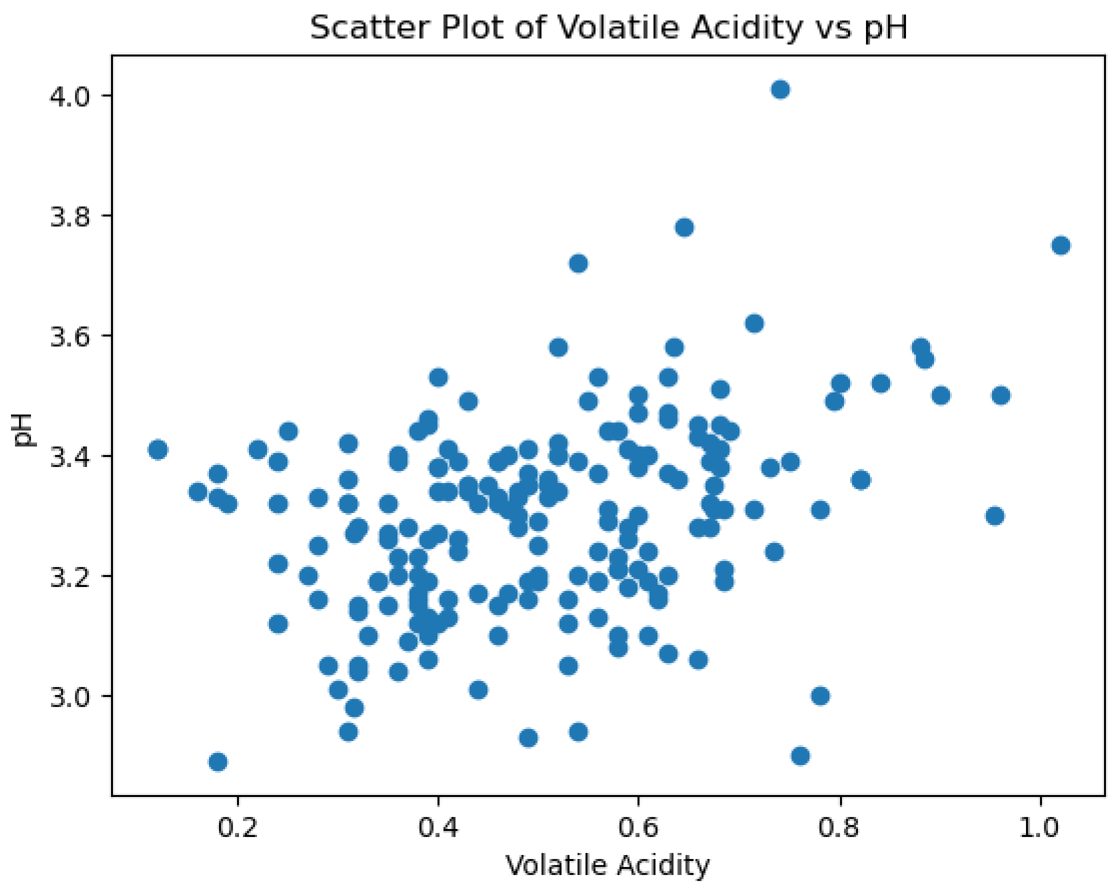
After reading the CSV file “A2data.csv” and creating a data frame object called winedf, I removed the missing values found in the data using dropna(). Following this, I identified the existence of 226 duplicate rows in the cleaned data. Although duplicated rows can introduce biases in analysis and inference, these duplicates could reflect multiple wine tasters consistently rating the same wine or multiple wines having the same production similarities. Therefore, I decided to retain these observations. I also performed sanity checks on the ‘pH’ column and ‘quality’ column, which revealed no invalid values, and then performed basic data exploration to understand the data better, which showed a relatively imbalanced dataset for the ‘quality’ column.

Task 1: Regression

Relationship Visualisation

Figure 1.1 below visualises the relationship between ‘volatile_acidity’ and ‘pH’ using a scatter plot. There does not appear to be a strong linear relationship between ‘volatile_acidity’ and ‘pH’, as the points are fairly spread out across the plot. However, there is a vague upward pattern hinted at by the clustering of the data points, suggesting a weak positive correlation.

Figure 1.1 Scatter Plot of Volatile Acidity vs pH



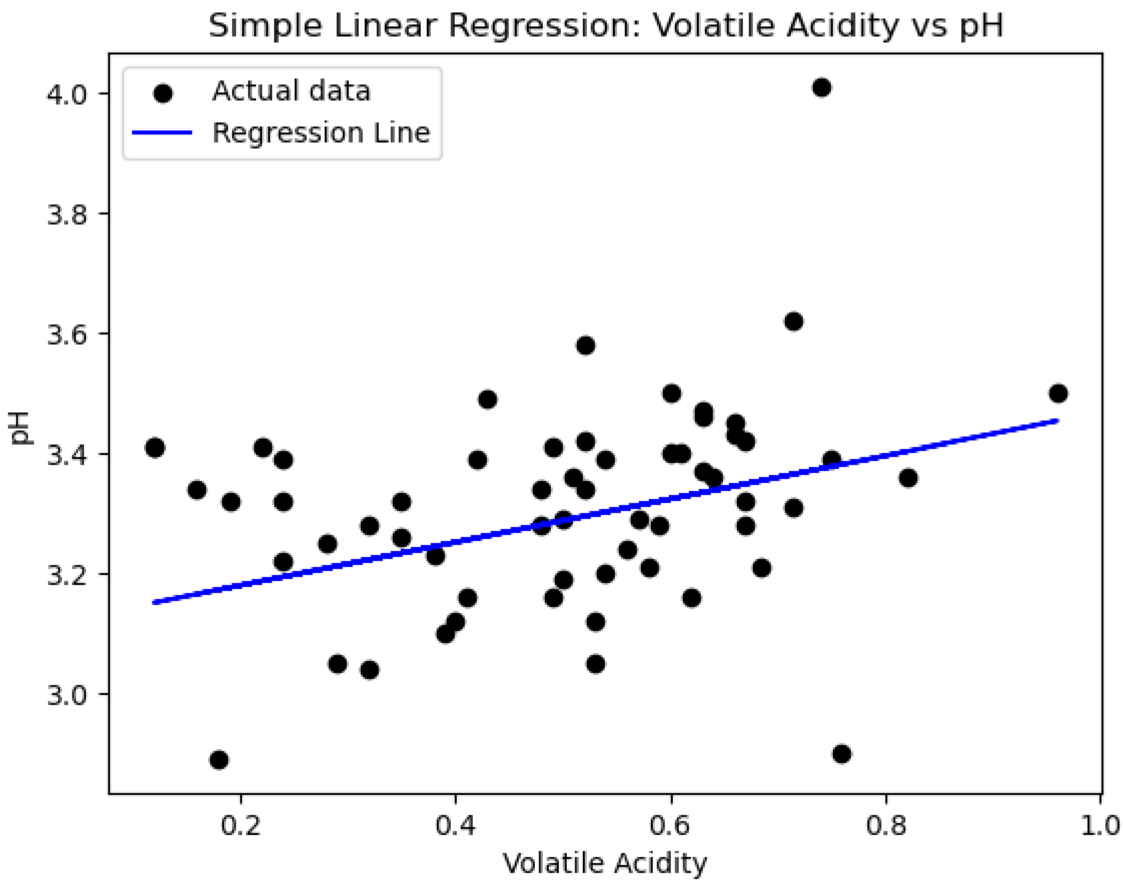
Simple Linear Model

To build the simple linear model, I first defined the independent variable as ‘volatile_acidity’ and the dependent variable as ‘pH’. Then, I split the sample data into training and test sets, using 70% of the data for the training set and the other 30% for the test set. Using training and test sets helps avoid overfitting and allows me to evaluate whether the model generalises well to unseen data. After fitting the linear regression model to the training data, the coefficients of the linear model received are 3.1071 for the intercept and 0.36 for the coefficient.

An intercept value of 3.1071 indicates that this is the expected value of ‘pH’ when ‘volatile_acidity’ is 0, and a coefficient value of 0.36 indicates a positive relationship. For every 1-unit increase in ‘volatile_acidity’, ‘pH’ increases by 0.36 units. However, this small value of the coefficient shows that this positive relationship is relatively weak. This validates the observation above about a weak positive correlation between ‘pH’ and ‘volatile_acidity’.

Next, I used the model to make predictions on the test data. To evaluate the model, I visualised the relationship between the predicted and actual test data by plotting the regression line on a scatter plot of the actual data points, shown below in Figure 1.2. The regression line has an upward trend, reflecting the positive relationship between ‘volatile_acidity’ and ‘pH’. However, the scatter of points spread out around the line indicate variability and that the correlation might not be very strong.

Figure 1.2 Scatter Plot of Volatile Acidity vs pH



To further evaluate the model, I calculated the following quantitative metrics measuring the performance of the linear regression model on the test data, shown in Table 1.1 below.

Table 1.1 Evaluation Metrics

R-squared (R2)	0.0405
Root Mean Squared Error (RMSE)	0.1686
Mean Absolute Error (MAE)	0.1273

Based on Table 1.1, an R-squared of 0.0405 indicates that the model does not fit the data well and that the variance in ‘pH’ is not explained well by ‘volatile_acidity’ (Shaibu, 2025). However, the RMSE of 0.1686 and MAE of 0.1273 are relatively small, indicating that the model’s predictions are, on average, close to the observed values. Even though RMSE and MAE appear relatively low and suggest that the errors in the model’s predictions are modest, the low R-squared value indicates that ‘volatile_acidity’ is not a strong predictor of ‘pH’.

Task 2: Classification

kNN

Before implementing a kNN (k-Nearest Neighbours) classifier, I first explored the target labels using the value_counts() method. The results show that there are 6 distinct target labels in the sample data, in which the majority class is the quality level of 5.0 (44% of the data), while the rarest class is 3.0, which only appears in 5 instances or 0.5% of the data. The distribution of the targets indicates that the dataset is relatively imbalanced. Following this, I separated the features of the data from the target class, then split them into training and test sets.

The next step was finding the optimal k value for the kNN model, which is the number of nearest neighbours the algorithm takes into account when classifying a new data point. To achieve this, k-fold cross validation was used to test a range of k values for kNN, consisting of all the odd numbers from 1 to 31. Only odd numbers were used to avoid ties, reduce bias toward majority classes in imbalanced datasets, and ensure the decision boundary falls between data points for more stable classification (European Information Technologies Certification Academy, 2023). Furthermore, a common rule of thumb is to choose k as the square root of the total number of points in the training data set (ScienceDirect Topics, n.d.). As there are 700 data points in the training set, this number is approximately 26. Hence, the range of 1 to 31 covers a suitable range which allows me to test the most flexible kNN model from k=1 to larger values which show the effects of more smoothing on the model’s performance.

The number of folds used in the cross-validation process followed the number of instances in the smallest class. In the function cross_val_score, if a value greater than the number of instances in the smallest class was employed in the cross-validation, it would risk leaving some folds without any instances of the smallest class, which may lead to inaccurate estimates.

After running the cross-validation test, the k-value which gives the highest cross-validation accuracy is k=17 (0.5386). Using this value of k, I set the kNN weight function to distance instead of the default uniform, so that closer neighbours have greater influence on classification than those further away, rather than all neighbours contributing equally. This adjustment may mitigate the biasedness towards majority classes due to imbalanced classes. The cross-validation accuracy score of this adjusted kNN model is higher than the uniformed kNN model, increasing to 0.5871 from 0.5386. Lastly, I adjusted p to 1, changing the power parameter to that used by the Manhattan Distance. This did not appear to improve the accuracy score, lowering it slightly from 0.5871 to 0.5843.

Hence, the final kNN model used to predict the test set has a k-value of 17 and utilises weighted distances, maintaining p at its default value of 2. To evaluate the model, a confusion matrix and classification report were produced based on the model's performance on the test set.

The confusion matrix shows that classes 5.0 and 6.0 tend to be confused with each other, although the model was still able to correctly predict them more times than incorrectly. Additionally, minority classes are predicted poorly. Class 7.0 is incorrectly predicted as 6.0 more than its correct class, and the rest of the classes, 3.0, 4.0 and 8.0 are not correctly predicted at all. In fact, the model does not predict any of the data points as 3.0 nor 4.0, and only predicts 8.0 once, incorrectly. Instead, all of its predictions centre around classes 5.0, 6.0 and 7.0, with a majority in 5.0 and 6.0. This suggests that there is majority class bias.

Additionally, the results from the classification report shows that class 5.0 has the highest recall of 0.65, meaning 65% of actual 5.0 samples were correctly labelled. Together with its precision of 0.58, the model appears to predict 5.0 reasonably well, similarly for class 6.0 which has slightly lower precision and recall than 5.0. Although class 7.0 has the highest precision of 0.6, it has a relatively low recall of 0.26 which signifies that 60% of all predictions labelled as 7.0 are correct, but only 26% of actual 7.0 samples were correctly predicted. However, the minority classes 3.0, 4.0 and 8.0 have 0 values for precision and recall, meaning all their samples were misclassified and that the model barely, if never predicted these classes.

Overall, the accuracy of this kNN model is 0.53, signifying that 53% of all test samples were correctly classified. However, this is not a good reflection of the model as the majority classes dominate 95% of the test set data. This high class imbalance allows the model to achieve apparently moderate accuracy, but in actuality, it completely fails to predict minority classes. Therefore, the macro average metric provides a more meaningful evaluation of the model as it treats minority and majority classes equally, avoiding biasedness towards majority classes (Geeks for Geeks, 2025a). This model has an overall macro averaged f1-score of 0.25, reflecting poor overall performance and exposing the model's struggle when taking into account minority classes.

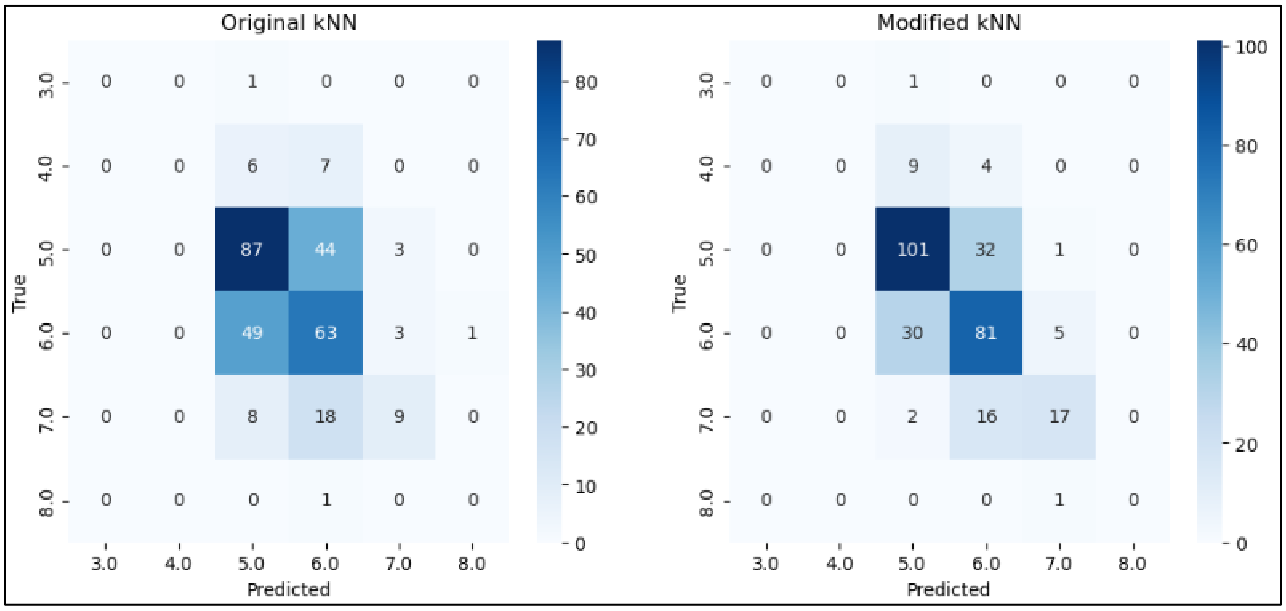
Modified kNN (and comparison)

A method to modify kNN for better performance is to perform hill climbing feature selection, in which subsets of features are iteratively tested to find the combination of features that improves the model performance. At the end of the process, the combination of features with the highest accuracy score is provided. Using the kNN model with k=17 and weights = distance from the previous section, the optimal combination identified from hill climbing feature selection consisted of 4 features, namely 'alcohol', 'sulphates', 'volatile_acidity' and 'pH', which combined resulted in an accuracy score of 0.6633.

I created a new set of training and test data using only the selected features to build the modified kNN model, fitting it onto this new set of training data and generating predictions from the new test set.

After generating the confusion matrix of the modified kNN, I plotted a side-by-side heatmap comparing the confusion matrices of the original and modified kNN model in Figure 2.1 below. As can be seen, the modified kNN predicts the classes 5.0, 6.0 and 7.0 correctly a higher number of times than the original kNN. Class 7.0 shows a significant improvement with the modified kNN as the number of points correctly predicted is now 17, compared to 9 with the original kNN. However, the modified kNN still performs poorly on the minority classes, failing to predict class 3.0, 4.0 and 8.0 correctly at all.

Figure 2.1 Heatmap Comparison of Confusion Matrices between Original and Modified kNN



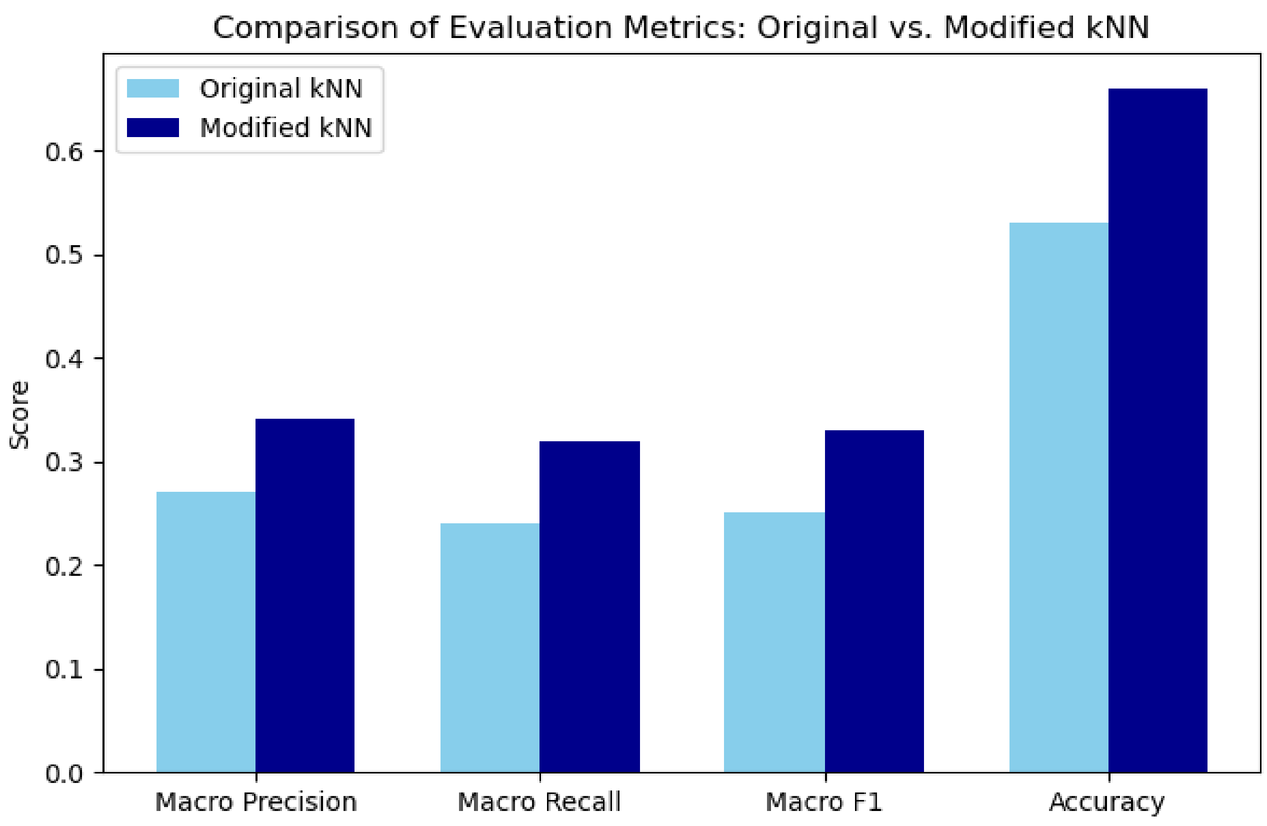
Original kNN (True \ Predicted)	3.0	4.0	5.0	6.0	7.0	8.0
3.0	0	0	1	0	0	0
4.0	0	0	6	7	0	0
5.0	0	0	87	44	3	0
6.0	0	0	49	63	3	1
7.0	0	0	8	18	9	0
8.0	0	0	0	1	0	0

Modified kNN (True \ Predicted)	3.0	4.0	5.0	6.0	7.0	8.0
3.0	0	0	1	0	0	0
4.0	0	0	9	4	0	0
5.0	0	0	101	32	1	0
6.0	0	0	30	81	5	0
7.0	0	0	2	16	17	0
8.0	0	0	0	0	1	0

The classification report of the modified kNN shows that class 5.0 has the highest f1-score, followed by 6.0 and 7.0, similar to the original kNN model. However, the modified kNN performs better across the board, with higher values for all metrics in all classes, except for the minority classes 3.0, 4.0 and 8.0, in which it performs the same as the original kNN, with no correct classifications. Overall, the modified kNN model has an accuracy of 0.66 and a macro average of 0.33 across all metrics, leading to the same conclusion as the original kNN. Although its accuracy score indicates moderately well performance, its macro average suggests majority class bias, and hence overall poor performance.

Figure 2.2 below illustrates the difference between macro-precision, recall, and f1-score, and overall accuracy between the original and modified kNN models. Only these metrics were selected for a visual depiction of the difference between the two kNN models as they provide a concise and meaningful comparison of overall model performance.

Figure 2.2 Comparison of Evaluation Metrics Between Original and Modified kNN



Overall, modified kNN produces higher scores for all the macro average metrics and accuracy than the original kNN, indicating improved performance. However, in both models, macro-averaged precision, recall and F1-score are significantly lower than the accuracy scores, exposing the weaknesses in both kNN models when it comes to predicting minority classes. Nonetheless, modifying kNN with hill climbing feature selection can lead to better performance because it ensures kNN uses only the most informative features. This helps the model focus on the most relevant attributes and reduces the impact of noisy attributes, which can distort the distance calculations in kNN and lead to misclassification.

Decision Tree (and Comparison)

To implement the decision tree classifier on the sample data, I first built a basic decision tree with untuned key parameters, leaving them in their default state (i.e. no max_depth, 2 min_samples_split, 1 min_samples_leaf etc.). After fitting the model on the training data and predicting the test set, the accuracy score obtained from this model was 0.5186, or approximately 52% accuracy. This moderate score does not indicate particularly strong predictive performance.

To find the optimal decision tree classifier, I tuned its key parameters, starting with the parameter max_depth. This parameter represents the maximum depth of the tree allowed, which helps ensure the model becomes more generalised and prevents it from memorising noise or outliers, thereby controlling overfitting. The range of depth values I selected to test consisted of 1 to 10, to see the effects of starting with a small maximum depth and incrementing gradually, and the default value of None. 10 was the chosen upper bound as this would have been the full tree depth which would create enough splits to assign each sample to its own leaf. The number 10 is derived from the approximate result of log2(N) in which N is 700, the number of samples in the training data. After iterating through the range of depth values, the value achieving the highest average cross-validation accuracy was 4, improving the score from approximately 52% to 58%.

The next parameter explored was min_samples_split, which represents the minimum number of samples required to split an internal node. This helps reduce overfitting by preventing overly-specific rules being created but which only apply to a few instances in the training data. The range of numbers I tested were 2, 5, 10, 15 and 20 as this provides a sufficient initial overview of the trend in the accuracy score. Using the decision tree classifier with a maximum depth of 4, the results demonstrate that the score does not improve when min_samples_split is increased from its default value of 2. It stays the same up to a minimum number of 5 samples, but gradually decreases once the minimum number reaches 10. Since the values 2 and 5 achieve the same score, I picked 5 as too small values for min_samples_split may cause overfitting.

I then tuned the parameter min_samples_leaf which represents the minimum number of samples required to be at a leaf node, which also helps control overfitting. The range of values I tested was from 1 to 20, with 1 being the default value for this parameter. Using a decision tree classifier model with max_depth = 4 and min_samples_split = 5, the results show that having a minimum of 5 samples required to be at a leaf node achieves the best cross-validation accuracy score, improving the model's prior score of 0.5786 to 0.5829.

The last parameter I tuned was max_features, the number of features the algorithm should consider when looking for the best split, which also helps control overfitting. The values I tested ranged from 1 to 6 as the data only has 6 features and this range sufficiently explores the effects of introducing randomness to using all available features. The default value of None was also included for testing. The results show that tuning max_features does not improve the accuracy score. Having 4, 5, 6 and None achieves the same score of 0.5829. As a result, I picked the smallest number out of these (4) to reduce the chance of overfitting to the training data.

Thus, the final decision tree classifier model used had the following adjusted parameters:

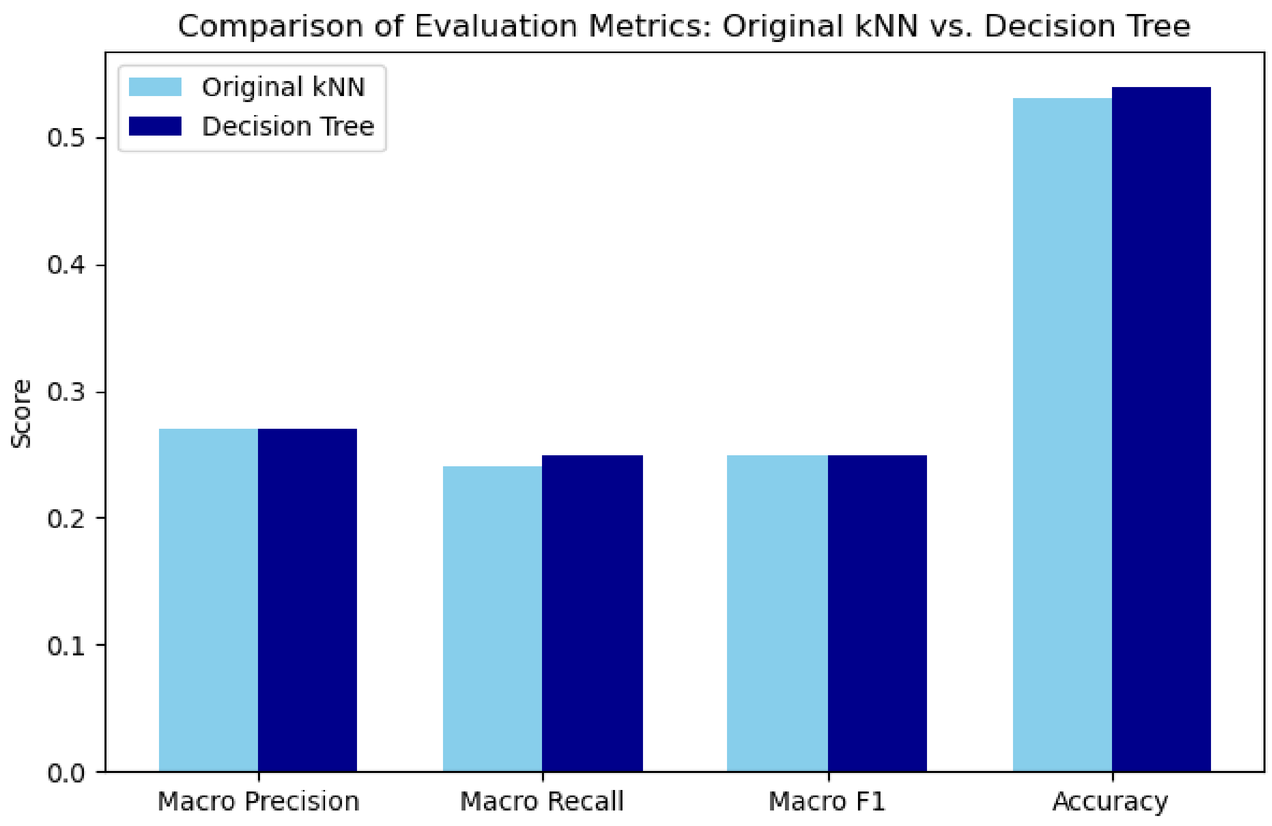
Table 2.1 Tuned parameters for final decision tree

Parameter	**Tuned value**
Max_depth	4
Min_samples_split	5
Min_samples_leaf	5
Max_features	4

The cross-validation accuracy score obtained by this model is 0.5829, an improved score compared to 0.5186 from the original, untuned model. Looking at the classification report on this model's performance, class 5.0 has the highest precision, recall and f1-score. Class 6.0 has a precision, recall and f1-score of 0.5, indicating that the model correctly predicts half of the instances it predicts as 6.0 and that it correctly identifies half of the actual instances for this class. Similar to the kNN models, class 3.0, 4.0 and 8.0 are not classified correctly and have 0 scores. Overall, this decision tree model has an accuracy of 0.54, but a significantly lower f1-score weighted macro average of 0.25, pointing to majority class bias. Hyperparameter tuning may have contributed to this bias as it reduces overfitting and enhances generalisation. By preventing the tree from memorising the training data, the model is less able to fit extremely small classes, which can lead to misclassification of minority-class samples.

Based on Figure 2.3 below, the decision tree's performance is relatively similar to that of the original kNN model, in terms of their overall performance. Both models have almost the same scores for the macro-averaged precision, recall and f1-score, as well as overall accuracy, with only minor decimal differences.

Figure 2.3 Comparison of Evaluation Metrics between Original kNN and Decision Tree



The similarity in performance between both models signifies that neither algorithm provides a substantial performance advantage for this dataset, and that both are similarly limited in their ability to correctly classify minority classes while being biased towards the majority classes, due to the sensitivity of kNN and decision trees to class imbalance. Nonetheless, both models have their strengths which may be useful in other situations. For instance, kNN is simple and makes predictions by looking at similar points, which works well if the data is clear and consistent. Moreover, it requires no assumptions about the data, and thus does not require the data to meet certain assumptions for it to be effective (MyEducator, n.d.). Meanwhile, decision trees can handle complicated patterns in the data and inherently reduces dimensionality by selecting the most important features (Great Learning Editorial Team, 2025).

Task 3: Clustering

k-Means

After separating the features and target column from the sample data, I started with exploring the impact of the number of clusters (k) on the performance of the k-Means algorithms using multiple clustering metrics, namely the Silhouette Score, Davies-Bouldin Index, Adjusted Rand Index (ARI) and Mutual Information Score (MI) (Geeks for Geeks, 2025b). Considering multiple evaluation metrics in this way provides a more comprehensive understanding of the clustering performance.

The k-Means algorithm was implemented using k values ranging from 2 to 10. The resulting evaluation metrics show that increasing the number of clusters decreases the Silhouette Score, suggesting that the clusters become less well-defined. A higher number of clusters may result in capturing noise rather than true patterns, leading to more scattered data points within clusters. The Davies-Bouldin Index is lowest for the smallest k values, 2 and 3. This indicates better and clearer clusters compared to the models with higher number of clusters. The Adjusted Rand Index (ARI) measures how accurate the clustering result is compared to the true labels, with a higher score signifying higher accuracy. In this case, all the ARI values are negative for all the tested k values. This means the model performs worse than random regardless of the number of clusters, indicating very poor clustering. Additionally, the Mutual Information (MI) score is low across all k values, bordering 0. This signifies that there is very little or close to no shared information between the true labels and the clusters assigned by the algorithm. Overall, the k-Means algorithm does not perform well on the sample data, and the clusters barely capture the true label structure.

Nonetheless, if I had to pick a k value for the algorithm, I would select k=6 despite the better performance of k=2 and k=3 from the evaluation metrics. This is because I know that the dataset contains only 6 observed classes. Choosing this value helps k-Means roughly match the number of true labels present in the data. Moreover, it is plausible that the reason for the better performance in the internal metrics (Silhouette Score and Davies-Bouldin Index) found with k=2 and k=3 may be due to the presence of 3 majority classes in the data. These internal metrics do not compare the clustering results to the true labels. On the other hand, ARI and MI are external metrics which evaluate against the true labels, and the ARI and MI results for k=2 and k=3 illustrate that their performance against the known 6 labels are weak like the other k-values. This suggests that k-Means may have aligned with the dominant structures in the data but not the actual class labels for k=2 and k=3. Thus, I decided not to go with the k values of 2 and 3.

The k-Means algorithm's performance when k is 6 has a silhouette score of 0.4343, which is close to 0, indicating that the sample is very close to the decision boundary between two neighbouring clusters. It has a Davies-Bouldin Index of 0.8652, which is close to 0 and suggests that the clusters are relatively tight and well-separated. However, the ARI of -0.0074 and Mutual Info Score of 0.0694 signify the poor performance of the model against the true labels.

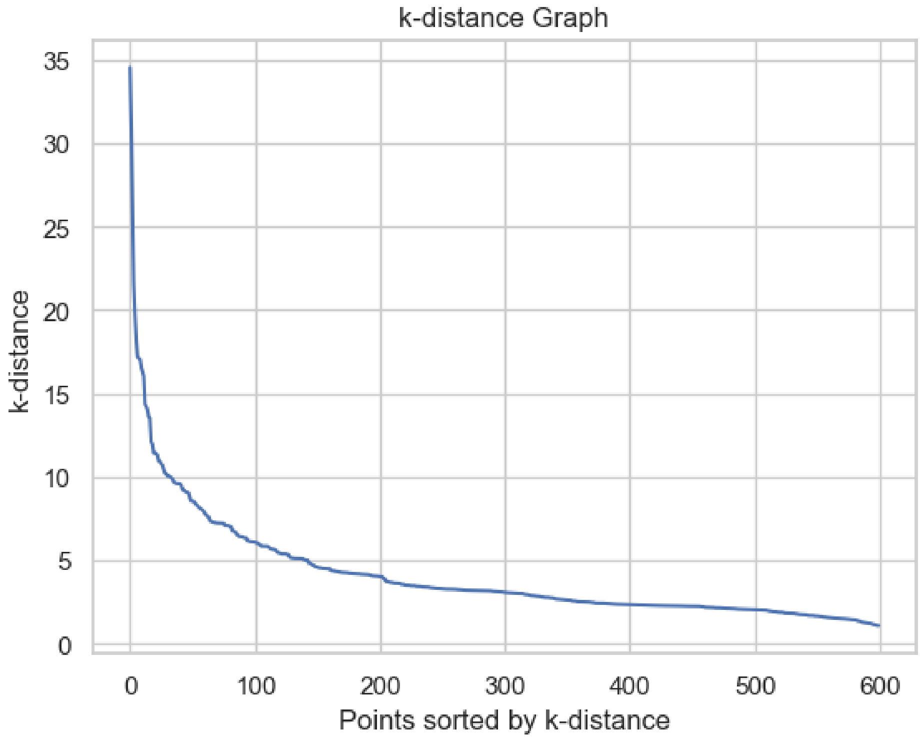
Overall, k-Means presents many limitations. It assumes that clusters are round or spherical and are around the same size. However, it is likely that the wine dataset is a different shape, and in addition, has extremely imbalanced classes. K-Means' struggle to handle such irregular clusters is evident in its poor clustering performance indicated by the ARI and MI scores. Another limitation of k-Means is its dependence on selecting an appropriate number of clusters (k), which is not always straightforward and may greatly affect results. Furthermore, k-Means is sensitive to outliers which may distort the clustering process. Extreme values in the data may have significantly affected k-Means' performance. A possible solution is to use an alternative clustering model like DBSCAN which can handle outliers and more irregular shaped data. Additionally, outlier handling techniques could be applied to reduce the impact of extreme values.

DBSCAN (and comparison)

To implement the DBSCAN algorithm, I begin by choosing an appropriate value for the epsilon parameter (Eps), and the MinPts parameter. These parameters are critical as Eps defines the radius of the neighbourhood, ultimately determining the size and shape of clusters. Meanwhile, MinPts determines how many numbers are needed in a neighbourhood for a point to be a core point.

A general guideline for selecting minPts is for the minimum value of minPts to be minPts >= D + 1, where D is the number of dimensions in the data set. As the data has 6 features, an appropriate minPts value could be 7. Based on this value, I calculated and plotted the k-distance graph using the distance to the 7th nearest neighbour, illustrated by Figure 3.1 below.

Figure 3.1 k-Distance Graph to Determine Optimal Eps

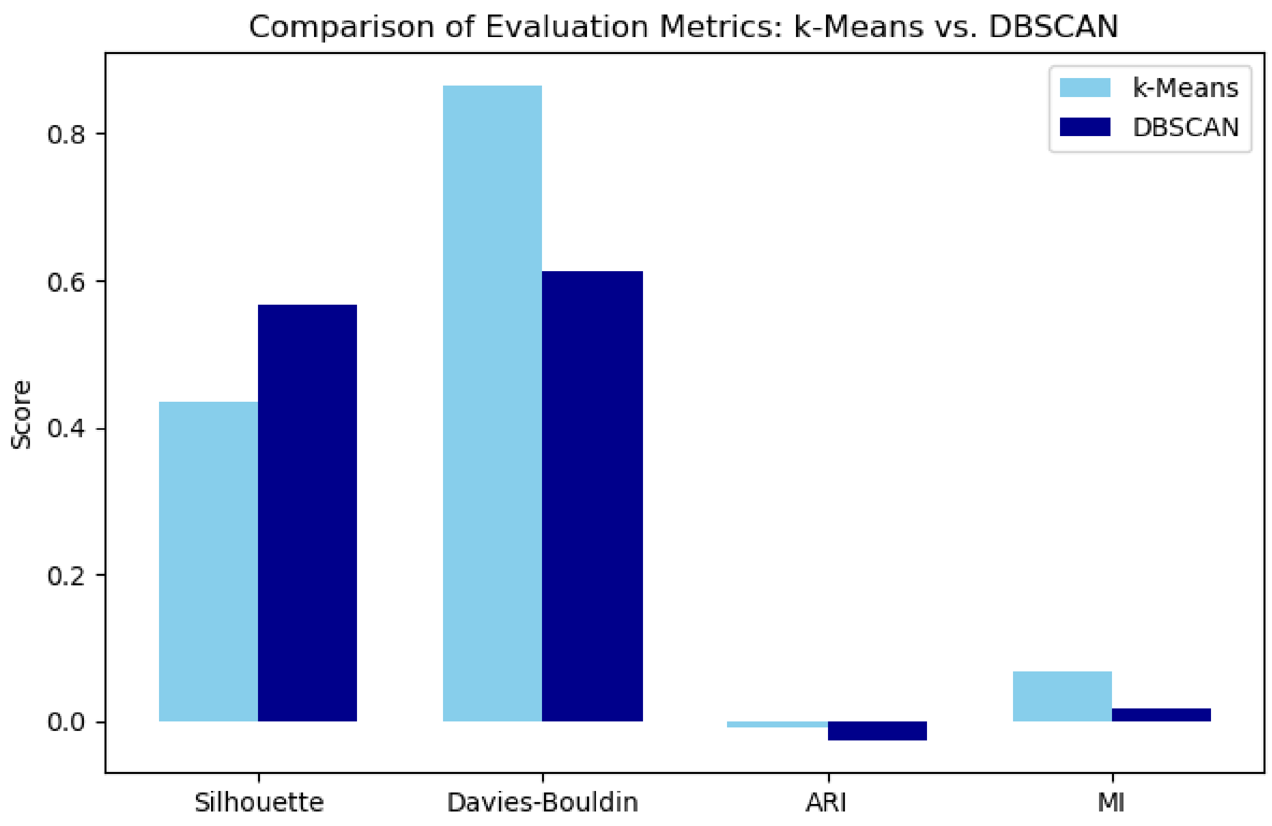


A k-distance graph was used to help select an appropriate value of Eps as it can help identify the point where dense regions are separated from sparser areas. Choosing an Eps near this point ensures that DBSCAN forms meaningful clusters while minimising noise points. Based on Figure 3.1, an appropriate Eps value is around 5 to 10 where the graph shows a strong bend.

With minPts = 7, I then explored the evaluation metrics of these Eps values. Based on the results, Eps=6 appears to be the most appropriate choice as it has the highest Silhouette Score (0.5662) and the lowest Davies-Bouldin Index (0.6125) out of the tested values. However, its performance against the true values as measured by the ARI (-0.0253) and MI (0.018) indicate very poor clustering performance, similar to the other Eps values used.

Compared to the k-Means with k=6, DBSCAN with Eps=6 and minPts=7 appears to perform worse. Although it has a higher Silhouette Score and lower Davies-Bouldin Index indicating better separation between clusters, it has a lower ARI and MI. This signifies that against the true labels, DBSCAN's clustering performed more poorly. Overall, both k-Means and DBSCAN performed inadequately on the data, with both having negative ARI scores and very low MI scores.

Figure 3.2 Comparison of Evaluation Metrics between k-Means and DBSCAN



DBSCAN likely performed worse than k-Means because it relies on density to form clusters. Very small classes may be treated as noise, and clusters with varying densities may be split or merged incorrectly. In contrast, k-Means assigns every point to a cluster. In this case, I used the exact number of features (6) as the number of clusters in the k-Means model. This may have led to slightly better performance than DBSCAN as k-Means may have better captured the smaller classes despite its weaknesses.

References

European Information Technologies Certification Academy. (2023, August 7). *Why is it recommended to choose an odd value for K in K nearest neighbors?* https://eitca.org/artificial-intelligence/eitc-ai-mlp-machine-learning-with-python/programming-machine-learning/introduction-to-classification-with-k-nearest-neighbors/examination-review-introduction-to-classification-with-k-nearest-neighbors/why-is-it-recommended-to-choose-an-odd-value-for-k-in-k-nearest-neighbors/

Geeks for Geeks. (2025a, June 9). *Macro average vs Weighted average*. https://www.geeksforgeeks.org/machine-learning/macro-average-vs-weighted-average/

Geeks for Geeks. (2025b, July 23). *Clustering Metrics in Machine Learning*. https://www.geeksforgeeks.org/machine-learning/clustering-metrics/

Great Learning Editorial Team. (2025, April 8). *Decision Tree Algorithm in Machine Learning*. Great Learning. https://www.mygreatlearning.com/blog/decision-tree-algorithm/#advantages-of-decision-trees

MyEducator. (n.d.). *Advantages and disadvantages of KNN*. https://app.myeducator.com/reader/web/1421a/11/q07a0/

ScienceDirect Topics. (n.d.) *K nearest neighbor*. ScienceDirect. https://www.sciencedirect.com/topics/immunology-and-microbiology/k-nearest-neighbor

Shaibu, S. (2025, March 12). *Linear Regression in Python: Your Guide to Predictive Modeling*. DataCamp. https://www.datacamp.com/tutorial/linear-regression-in-python