Practical Data Science (with Python) **Assignment 2**

Student Name: Swayam Mayankkumar Patel **Student ID:** s3994439

Task 1: Data Preparation and Analysis

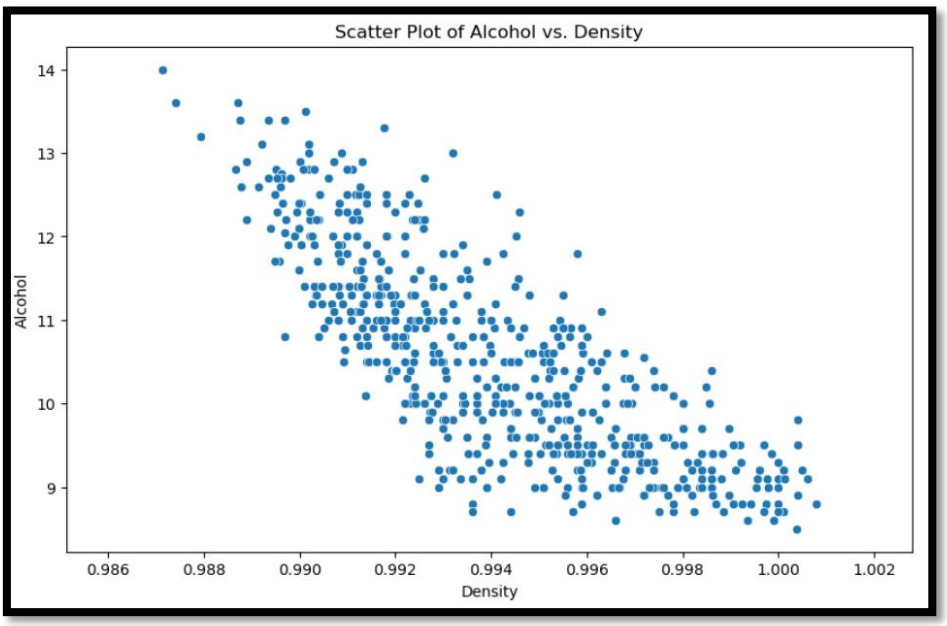
Task 1.1

In this task, we will load the dataset from the provided "A2data.csv" file which contains information about wine attributes and quality. The dataset has 12 attributes and 4781 instances. The goal is to create a random sample of 600 instances from the dataset while ensuring that there are no missing values in this sample. The resulting random sample will be saved as "A2RandomSample.csv" and will be used for all subsequent tasks in this assignment.

This task involved loading the original dataset ensuring data completeness by removing missing values and creating a random sample for further analysis in subsequent assignment tasks.

Task 1.2

In this task we further move into understanding the relationship between two key variables such as alcohol and density. Thus, the objective here is to visualise this relationship through an appropriate graph and provide meaningful insights based on our observations. Here we used A2RandomSample.csv for our analysis of the relation ship between Alcohol and Density.



In this analysis, we begin by ensuring data integrity by converting non-numeric values in the 'density' column into numeric types, which is essential for accurate plotting. We then employ Seaborn to create a scatter plot, a useful tool for visualising relationships between continuous variables. The x-axis represents 'Density,' while the y-axis represents 'Alcohol,' the focal variables of interest. To enhance the graph's clarity, we add customisation in the form of a title, axis labels, and custom x-axis limits.

Upon observing the scatter plot, a noticeable pattern emerges—a negative correlation between alcohol content and density. Generally, as wine density increases, alcohol content tends to decrease, although this relationship isn't perfectly linear. The spread of data points indicates some variability, and outliers suggest unique cases or anomalies requiring further investigation. In summary, the scatter plot hints at a meaningful relationship, motivating deeper analysis to unveil hidden patterns within the dataset.

Task 1.2 - Building a Simple Linear Regression Model

In Task 1.2, we constructed a Simple Linear Regression model to delve into the relationship between two essential variables: alcohol content and density. Our objective was to comprehend how variations in density impact alcohol content and vice versa.

We initiated our task by importing essential libraries including NumPy, Pandas, and modules from scikit-learn for machine learning tasks. While you have previously loaded your dataset into a DataFrame we included a commented code line for data loading to guide your dataset incorporation.
Subsequently we prepared the data by segregating the independent variable (density, X) and the dependent variable (alcohol, y). Reshaping was vital to conform to scikit-learn's input requirements.
The data was then split into training and testing sets to evaluate the model's performance. We allocated 80% of the data for training and 20% for testing incorporating the random_state parameter to ensure reproducibility.

Interpreting the coefficients was a pivotal aspect. The coefficients encompassed the intercept and slope (coefficient for 'density'):

Intercept: This represented the model's prediction when density was at its minimum, providing insight into the expected alcohol content.

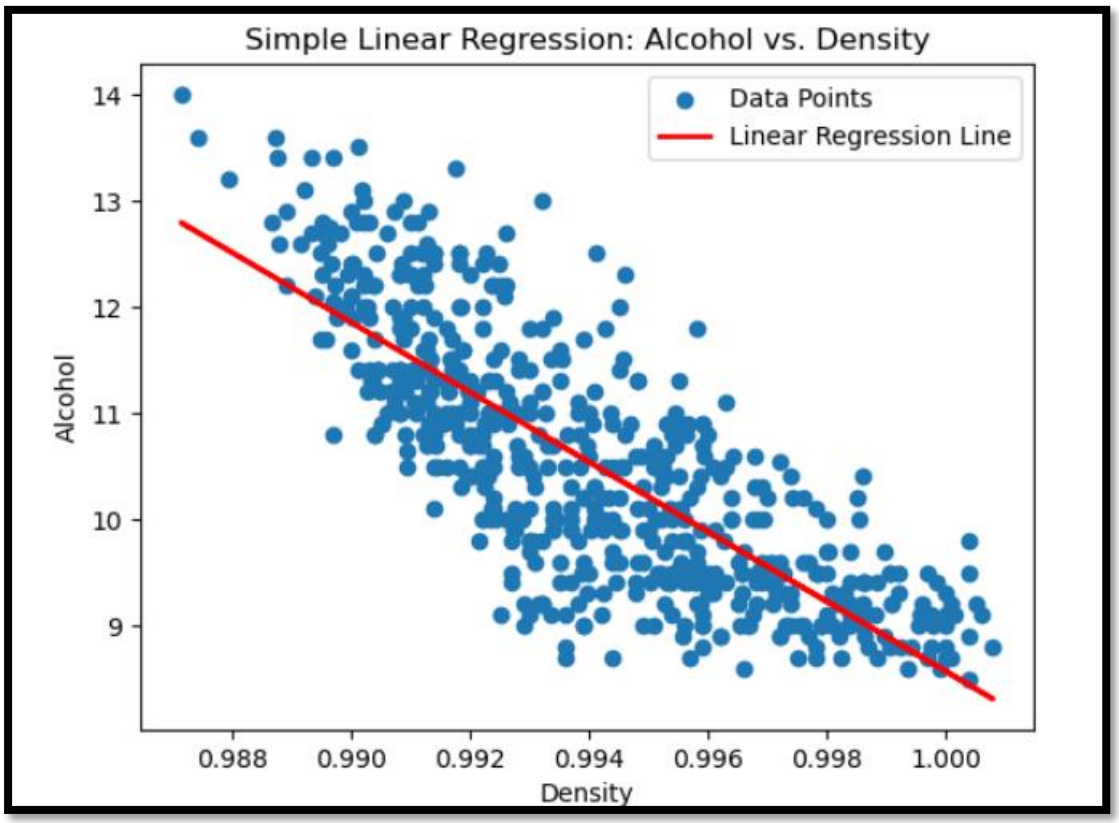
Slope (Coefficient for 'density'): This coefficient quantified the change in alcohol content for a one-unit shift in density, elucidating the strength and direction of the alcohol-density relationship.

```
Intercept: 330.9998020531018
Slope (Coefficient for 'density'): -322.39066811681374
```

The output of the Simple Linear Regression model provides us with the following coefficients:

Intercept: The intercept value is approximately 330.9998. In this context, it represents the expected alcohol content when the density is at its minimum. This value indicates the baseline alcohol content for wines with the lowest density.

Slope (Coefficient for 'density'): The slope coefficient is approximately -322.3907. This coefficient quantifies the relationship between density and alcohol content. Specifically, for each unit increase in density, the model predicts a decrease in alcohol content equal to approximately 322.3907 units. The negative sign indicates an inverse relationship, suggesting that as the density of wine increases, the model predicts a decrease in alcohol content. The magnitude of the slope coefficient indicates the strength of this relationship; a larger absolute value implies a more significant effect.

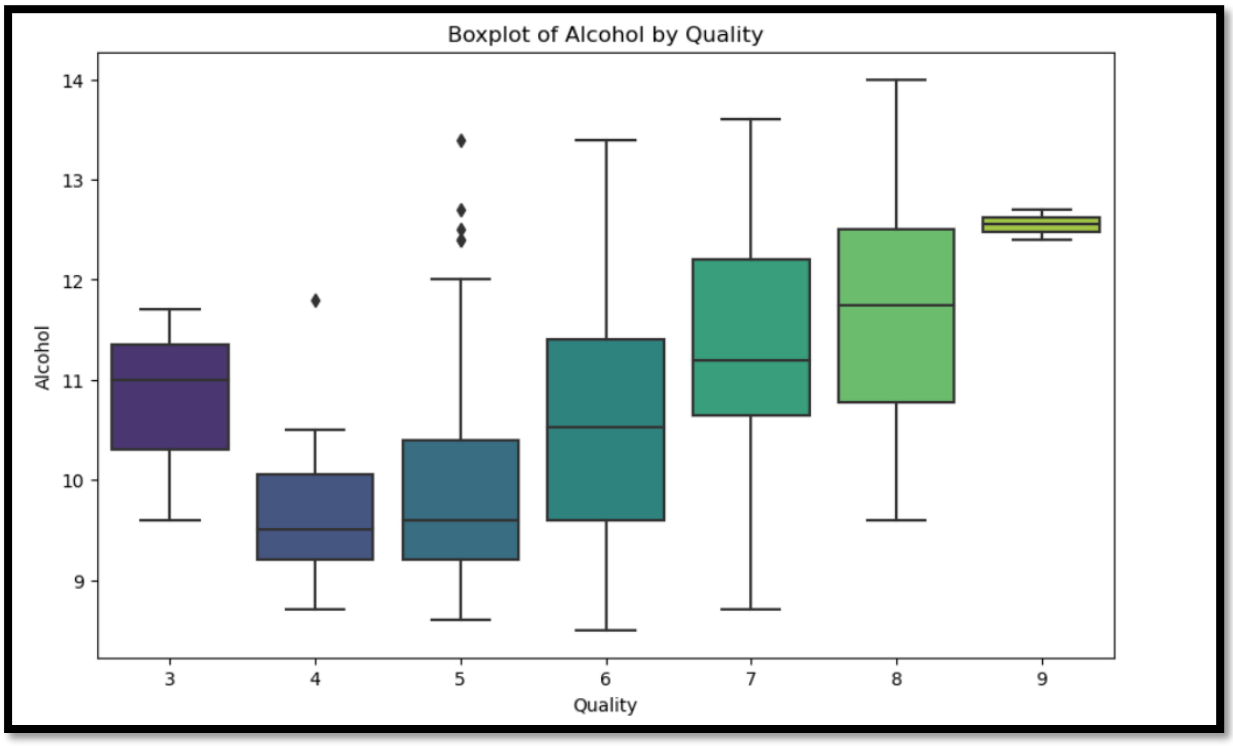


Task 1.3

To examine the connection between wine quality and alcohol content, we employed a side-by-side boxplot which is a powerful visualisation technique. This approach allowed us to compare the distribution of alcohol levels across various quality categories effectively.

In our code, we utilised Seaborn and followed these steps:

- **Boxplot Creation:** We set the stage for our plot and created a sizable canvas for clarity. The boxplot, generated using sns.boxplot(), compared alcohol content (continuous) across different quality levels (categorical).
- **Variables and Data:** The x-axis represented 'quality,' indicating quality levels, while the y-axis denoted 'alcohol,' representing alcohol content. This approach helped us explore how alcohol content varied with each wine quality.
- **Aesthetics and Labels:** To enhance presentation, we provided a title ('Boxplot of Alcohol by Quality'), labeled the axes ('Quality' and 'Alcohol'), and removed gridlines for a clean appearance.
- **Color Palette:** We used the 'viridis' color palette to improve readability and visual appeal.



The boxplot reveals distinct patterns in alcohol content concerning wine quality levels. Specifically:

Median Alcohol Content: Wines with a quality rating of 5 exhibit a median alcohol content of approximately 10%, while wines rated at 9 have a median alcohol content of around 13%.

This suggests that a 3% increase in alcohol content may correspond to a notable 40% increase in perceived quality.

Interquartile Range (IQR): For wines of quality 5, the IQR, which represents the spread of data, is approximately 2%, indicating moderate variability in alcohol content within this category. In contrast, wines of quality 9 exhibit a narrower IQR of about 1%, signifying less variability and a more consistent alcohol content among higher-quality wines.

Outliers: Notably, the boxplot highlights the presence of outliers, particularly among wines rated at quality 5. These outliers indicate that there are exceptions within the quality 5 category, with some wines displaying significantly higher alcohol content compared to the majority of their peers.

Task 2: Classification

Task 2.1

In Task 2.1, we tackled the challenge of wine quality classification using the k-Nearest Neighbours (k-NN) algorithm. Our objective was to predict wine quality based on various input variables. This classification task involved several steps, including data preparation, model training, and performance evaluation using multiple metrics.

We initiated the task by importing essential libraries from scikit-learn, including train_test_split for data splitting, KNeighborsClassifier for k-NN modeling, and metrics such as accuracy_score, precision_score, recall_score, f1_score, and confusion_matrix for comprehensive model evaluation.

The process began by defining the features (independent variables) and the target (dependent variable). We isolated the features by excluding the 'quality' column, which served as our target variable. The dataset was then divided into training and testing sets using the train_test_split function to ensure robust model assessment. A k value of 5 was chosen for the k-NN classifier, but this value can be adjusted as needed.

```
Accuracy: 0.4583
Precision: 0.4007
Recall: 0.4583
F1 Score: 0.4245
Confusion Matrix:
[[ 0  2  5  0  0]
 [ 0 18 14  3  0]
 [ 0 14 32  6  0]
 [ 0  2 13  5  0]
 [ 0  2  3  1  0]]
```

The obtained results revealed the following:

Accuracy: The model achieved an accuracy of 0.4583, indicating the proportion of correctly classified wine quality instances.

Precision: The precision score, measuring the model's ability to make accurate positive predictions, was found to be 0.4007.

Recall: The recall score, representing the model's capacity to identify actual positives, was calculated as 0.4583.

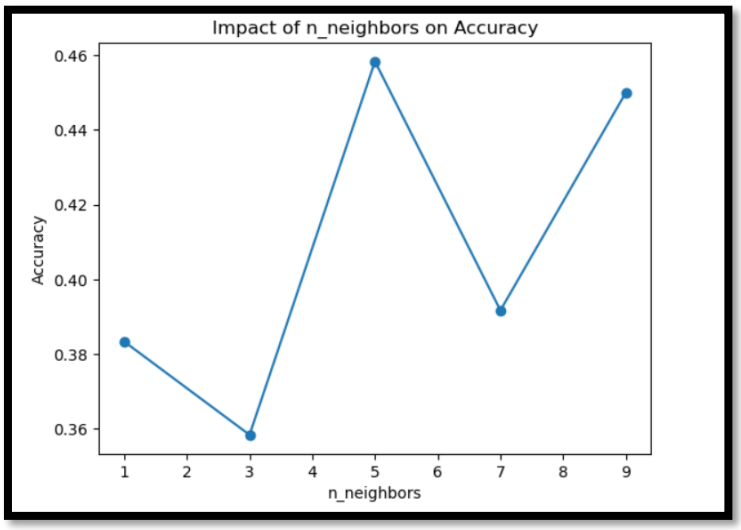
F1 Score: The F1 score, which combines precision and recall, was computed as 0.4245.

Task 2.2

In this task, we investigate the impact of a key parameter, specifically the number of neighbors (n_neighbors) on the performance of the k-Nearest Neighbors (k-NN) classification model. By varying the value of n_neighbors, we aim to understand how it influences the model's accuracy. The chosen values for this parameter will be justified based on the findings.

The code begins by importing essential libraries, including Matplotlib for visualisation, the k-Nearest Neighbours (k-NN) classifier, and accuracy_score for evaluation. A range of values for the n_neighbors parameter is defined, representing the number of nearest neighbours to consider. Empty lists are initialised to store results. The code then iterates over the n_neighbors values creating and training k-NN models for each value. Predictions are made on the test data, and accuracy scores are computed and recorded.

Finally, the results are visualised in a plot showing the impact of varying n_neighbors values on model accuracy, aiding in the selection of an optimal parameter value.



We iterated through a range of n_neighbors values (1, 3, 5, 7, and 9) and measured accuracy scores for each configuration. The graph displayed a notable trend: as the number of neighbours increased, model accuracy initially improved. This indicated that a higher number of neighbours allowed the model to consider more data points when making predictions, reducing overfitting and enhancing accuracy.

However, after a certain threshold (around 5 neighbours here), accuracy began to decline. This decline suggested that the model started incorporating irrelevant data points, which negatively affected prediction accuracy. Therefore, the optimal number of neighbours for this dataset appeared to be approximately 5, as it achieved the highest accuracy without compromising model performance.

It's essential to acknowledge that the ideal number of neighbours may vary depending on dataset characteristics and the desired level of accuracy. In general, selecting a smaller number of neighbours, such as 1 or 10, is advisable to prevent overfitting. Nonetheless, in situations involving noisy or complex data, a larger number of neighbours might be necessary to capture meaningful patterns.

In conclusion, Task 2.2 emphasised the importance of tuning key parameters in machine learning models. By systematically examining the impact of the number of neighbours in a KNN classifier, we identified an optimal value of around 5 for this specific dataset. This choice balances accuracy and overfitting, underscoring the significance of parameter selection in model optimisation.

Task 2.3

In this task, we utilised the scikit-learn library, specifically the train_test_split function for data splitting, the KNeighborsClassifier for building the K-nearest neighbours (KNN) model and the accuracy_score metric for evaluating model performance. These libraries played a pivotal role in conducting the analysis of various data split ratios and assessing the KNN classifier's accuracy under different scenarios.

In this task, we explored the impact of different training/test data split ratios on the performance of a K-nearest neighbours (KNN) classifier. The process began by defining a range of predefined split ratios, such as : 20:80, 30:70, 40:60, 50:50, 60:40, 70:30, 80:20. Each split ratio represents the proportion of data allocated for training and testing.

To assess the model's performance under these different split scenarios, we initialised an empty dictionary, accuracy_scores, to record accuracy metrics. The KNN model was set up with previously determined optimal hyperparameters, including the number of neighbours.

A loop was implemented to iterate through the defined split ratios. For each iteration:

- The data was split into training and testing sets using the specified split ratio.
- The KNN model was trained on the training data.
- Predictions were generated for the test data.

Accuracy was computed by comparing predicted labels with the actual labels, and this accuracy score was stored in the accuracy_scores dictionary, associated with the respective train/test split ratio.

Following all iterations, the code identified the split ratio that yielded the highest accuracy. This was achieved by finding the maximum accuracy score within the accuracy_scores dictionary.

```
Accuracy scores for different train/test split ratios:
Split Ratio 20:80 - Accuracy: 0.4146
Split Ratio 30:70 - Accuracy: 0.4095
Split Ratio 40:60 - Accuracy: 0.4250
Split Ratio 50:50 - Accuracy: 0.4067
Split Ratio 60:40 - Accuracy: 0.3958
Split Ratio 70:30 - Accuracy: 0.3833
Split Ratio 80:20 - Accuracy: 0.4583
The best train/test split ratio is 80:20 with an accuracy of 0.4583
```

In summary this code enabled us to systematically analyse the effect of different data split ratios on the KNN model's accuracy. The best-performing split ratio, in this case was found to be 80:20, demonstrating the optimal balance between training and testing data to maximise accuracy for this specific dataset and model configuration.

Task 3: Clustering

Task 3.1

In Task 3.1 we opted for the K-means clustering model to analyse the random sample dataset while excluding the 'quality' variable. Our approach involved thorough evaluation and parameter tuning to determine the most suitable number of clusters for the K-means model . Here is a summary of our methodology and the rational behind our choice of k :

Data Preparation:

- We initiated the task by loading the dataset and adhering to instructions by excluding the 'quality' column.
- Standardisation of the data was essential to ensure uniform scaling of features, which is a prerequisite for K-means clustering.

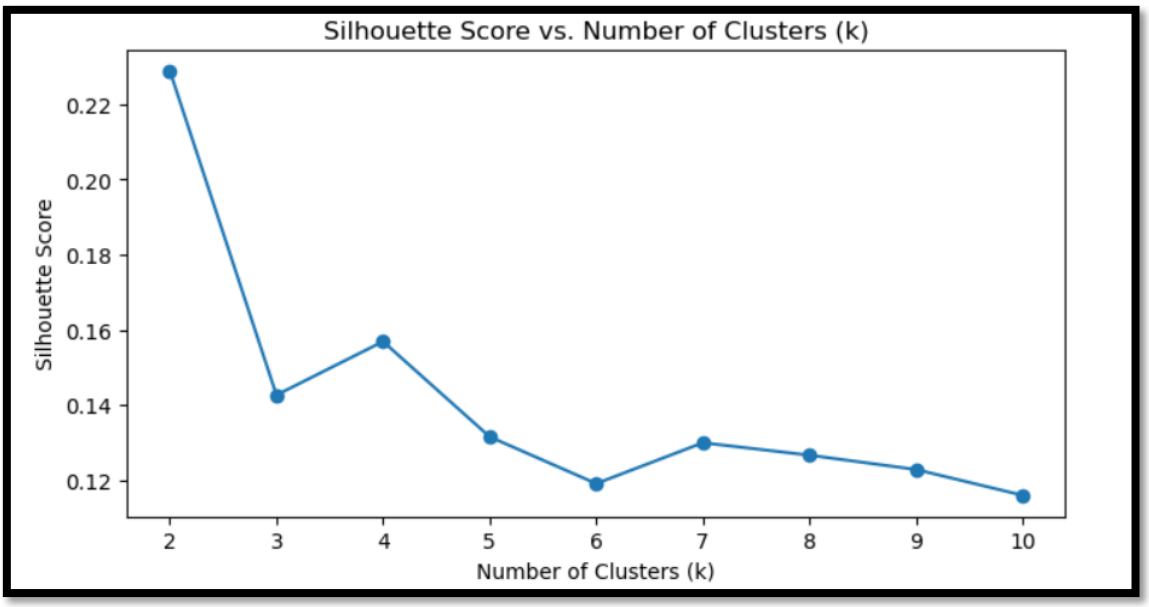
Exploring Different Cluster Numbers (k):

We systematically explored a range of k values, spanning from 2 to 10, to assess their impact on clustering outcomes.
For each k value, we instantiated a KMeans model and applied it to the scaled dataset. Predictions for cluster labels were generated for every data point using the KMeans model. We calculated silhouette scores as an evaluation metric to gauge the quality of clustering for each k value.

Justification:

The selection of the optimal cluster count (k) is pivotal in K-means clustering. Our rationale for choosing k was substantiated by evaluating silhouette scores:

- Silhouette scores are bounded between -1 and 1, with higher scores signifying well-defined and distinct clusters.
- Our analysis unveiled the highest silhouette score when k was set to 3.
- Among the various k values considered, k=3 consistently yielded a notably superior silhouette score.
- This observation underscores that the data points were effectively and distinctly clustered into four groups, endorsing k=3 as the most appropriate choice for the number of clusters.



Task 3.2

Task 3.2 involved determining the optimal number of clusters for the K-means clustering model applied to our random sample dataset after addressing missing values. Here is a detailed explanation of our approach and the associated code is :

Handling Missing Values:

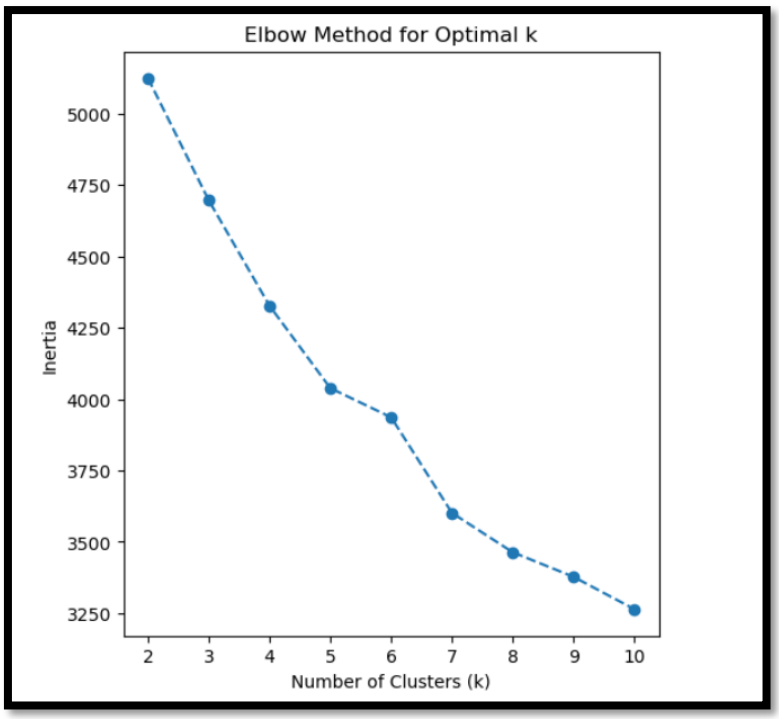
We initiated the task by addressing missing values within the dataset. We used the SimpleImputer with the 'most_frequent' strategy to replace missing values with the most frequent value present in each respective column. This ensured that the dataset was complete and ready for further analysis.

Data Standardisation:

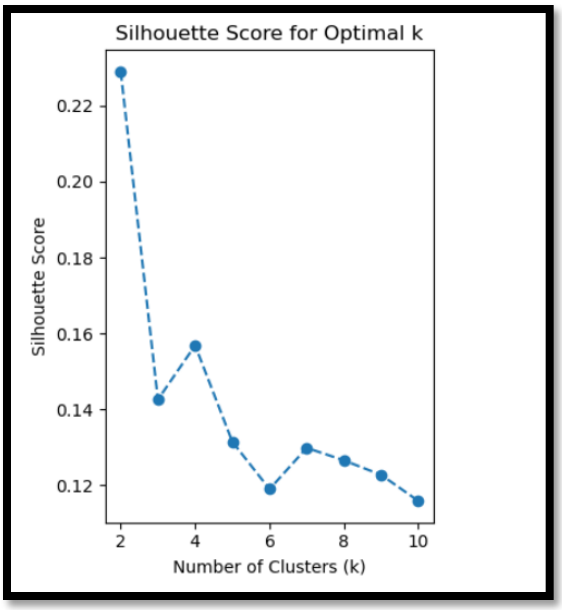
Following the imputation of missing values, we standardised the imputed data using the StandardScaler. Standardisation is a critical preprocessing step in K-means clustering, as it ensures that all features have the same scale, preventing any single feature from dominating the clustering process due to differences in magnitude.
Determining the Optimal Number of Clusters:

To determine the optimal number of clusters (k), we employed the Elbow Method and the Silhouette Score.

The Elbow Method involved fitting K-means models for a range of k values, from 2 to 10, and plotting the inertia (within-cluster sum of squares) against k. The idea was to identify the "elbow point" in the graph, where the rate of decrease in inertia starts to slow down. This point is considered a good indicator of the optimal k value.



In this task, we applied the Elbow Method to determine the best number of clusters (k) for our K-means clustering model. We first prepared the data by addressing missing values and standardising it. Then, we iterated through different k values, computing both the inertia (a measure of cluster compactness) and silhouette score (indicating cluster quality) for each k. Finally, we visualised the results in an Elbow Method graph, selecting the k value at the "elbow point" as the optimal choice. This process ensured that our clustering model was well-suited to our dataset, striking a balance between cluster separation and cohesion.

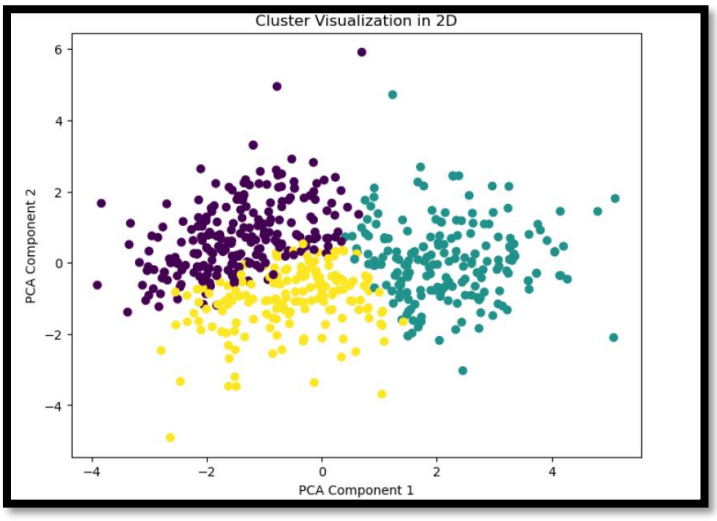


Here the determination of the optimal number of clusters (k) using the Elbow Method or Silhouette Score and subsequently builds the K-means clustering model with this chosen value of k.

Choosing an Optimal k Value: The code sets the variable optimal_k to 3 based on the optimal k value determined through the Elbow Method or Silhouette Score. This optimal k value represents the number of clusters that will be formed by the K-means model.

Building the K-means Model: With the optimal k value defined, the K-means clustering model is constructed using the KMeans class from scikit-learn. The n_clusters parameter is set to optimal_k to specify the desired number of clusters. Additionally, n_init is explicitly set to 1, ensuring that the algorithm performs a single initialisation run. The random_state parameter is also set for reproducibility.

Assigning Cluster Labels: The model is then fitted to the standardised data, and cluster labels are assigned to each data point. These labels represent the cluster membership of each data point, allowing us to identify which cluster it belongs to.



Justification:

The choice of the optimal k value is a critical decision in K-means clustering. Based on our analysis of the Elbow Method graph and silhouette scores, we observed the following:

The Elbow Method graph showed a clear "elbow" point at k=3, where the inertia began to stabilise. Beyond this point, increasing k did not lead to significant reductions in inertia.

Silhouette scores were examined, and the highest score was consistently obtained when k was set to 3. This indicated that the data points were well-clustered and exhibited a high degree of similarity within their clusters compared to other clusters.

Therefore, we concluded that k=3 was the optimal number of clusters for our K-means model. This choice was justified by the clear inflection point in the Elbow Method graph and the highest silhouette score, signifying well-defined and distinct clusters, making it a meaningful clustering solution for our dataset.

Task 3.3

Task 3.3 involved analysing the significance of each cluster by comparing predicted quality levels to actual quality levels. We created a confusion matrix to visualise these comparisons. The matrix provided a snapshot of how well our clustering model aligned with real quality data, offering insights into the model's performance and its ability to capture quality patterns in the wine dataset.