Classification of Red and White Wines Using Supervised and Unsupervised Learning Techniques

Melany Yohana Gomez Herrada

December 6, 2024

Abstract

This report analyzes different types of wines using various statistical learning methods. The main goal is to classify wines as red and white based on their chemical properties and to explore the data's underlying structure. For supervised learning, we applied models like logistic regression, K-Nearest Neighbors (KNN), decision trees, random forests, and XGBoost to predict the wine types. In unsupervised learning, we used techniques such as Principal Component Analysis (PCA), UMAP, and t-SNE to visualize the data and discover patterns without predefined labels. Clustering algorithms like K-Means and hierarchical clustering were used on the reduced datasets to group similar wines together. The study concludes by discussing the findings and highlighting key insights from both supervised and unsupervised methods.

Contents

List of Figures

List of Tables

1 Introduction

Wine classification plays a crucial role in the wine industry, aiding in quality control, authentication, and enhancing consumer experience. Traditionally, the differentiation between red and white wines relies on sensory evaluations, which can be subjective and inconsistent. By leveraging chemical properties, we can develop objective and reproducible methods for wine classification. The dataset used for this analysis originates from Portuguese "Vinho Verde" wines, which are known for their unique flavor profiles and are classified into red and white variants. The dataset contains several physicochemical properties, including acidity, alcohol content, sugar levels, and pH, which serve as input features for the predictive models. The results of this study demonstrate the potential of machine learning in predicting wine types using chemical characteristics alone. By identifying the most influential features and leveraging the strengths of various models.

2 Data

2.1 Dataset Source

Cortez, P., Cerdeira, A., Almeida, F., Matos, T., Reis, J. (2009). *Modeling wine preferences by data mining from physicochemical properties.* UCI Machine Learning Repository.
https://doi.org/10.24432/C56S3T

2.2 Dataset Description

The dataset compiled by Cortez et al. (2009) comprises two datasets related to red and white variants of the Portuguese "Vinho Verde" wine. The combined dataset includes a total of 6,497 samples, with 1,599 red wine samples and 4,898 white wine samples. Each sample is characterized by 11 physicochemical properties obtained through laboratory tests, along with a quality score assigned by wine experts based on sensory evaluation.

Table 1: Description of Variables in the Wine Dataset

Variable	**Description**
1. Fixed Acidity	Non-volatile acids (primarily tartaric and malic) in g/dm^3
2. Volatile Acidity	Acetic acid in g/dm^3; high levels can lead to an unpleasant vinegar taste
3. Citric Acid	Citric acid in g/dm^3; adds freshness and flavor to wines
4. Residual Sugar	Amount of sugar remaining after fermentation in g/dm^3
5. Chlorides	Chloride content in g/dm^3; represents the amount of salt in the wine
6. Free Sulfur Dioxide	Free form of SO_2 in mg/dm^3; acts as an antimicrobial and antioxidant
7. Total Sulfur Dioxide	Total SO_2 in mg/dm^3; includes both free and bound forms
8. Density	Density of the wine in g/cm^3; influenced by sugar and alcohol content
9. pH	Measure of acidity or alkalinity; typical values range from 2.9 to 3.9
10. Sulphates	Potassium sulphate content in g/dm^3; contributes to SO_2 levels and wine stability
11. Alcohol	Alcohol content by volume (% vol.)
12. Quality	Quality score between 0 and 10; based on sensory evaluation by wine experts

3 Data Preparation

The dataset underwent several preprocessing steps, below, these transformations are detailed:

Combining Datasets and Adding Target Variable

- **Merging Red and White Wine Datasets:** The red and white wine datasets were combined into a single dataset to facilitate comprehensive analysis.

Handling Missing Values

- **Missing Data Assessment:** The combined dataset was examined for missing values using `is.na()` function. The assessment revealed no missing values, ensuring data completeness.

Removing Duplicate Entries

- **Duplicate Row Elimination:** Duplicate entries were identified and removed using the `unique()` function. This step is critical to prevent biases in the analysis that can arise from overrepresented data points. There were no duplicates in the data.

Encoding Categorical Variables

- **Binary Encoding of `type`:** The `type` variable, originally containing the string values `"red"` and `"white"`, was converted to numeric format with `1` for red wine and `0` for white wine.This binary encoding serves as the target variable for classification tasks.

Normalization of Features

- **Scaling Numeric Variables:** All numeric features were normalized to a range between 0 and 1 using min-max scaling. Normalization standardizes the data scale across variables, mitigating scale-induced bias and improving the performance of algorithms sensitive to variable scaling.

These preprocessing steps ensured that the dataset was well-prepared for both supervised and unsupervised analysis, enhancing the reliability and validity of the analytical outcomes.

4 Descriptive Statistics

To the data preparation process was followed by various descriptive statistics, the descriptive goal is to elucidate the relationships among various physicochemical properties of red and white wines and their correlation with wine type.

4.1 Frequency Distribution Target Variable

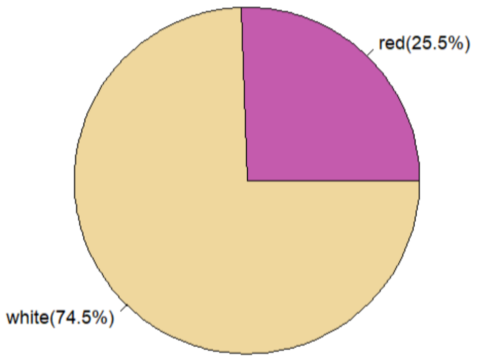


Figure 1: Distribution of the Variable "Type"

4.2 Correlation Analysis

The correlation analysis helps identify redundant features, reduce multicollinearity, and understand which variables are most influential, which is valuable for feature selection and model building.

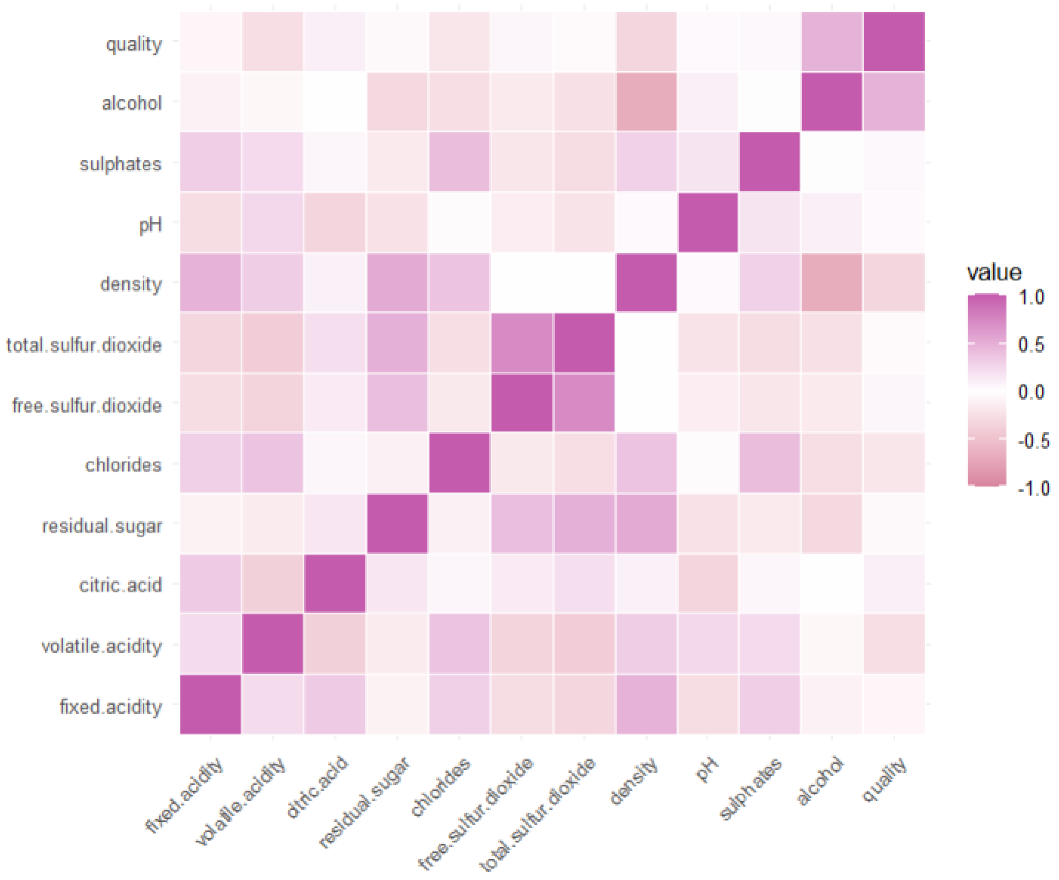


Figure 2: Heatmap of the Correlation Matrix

The correlation matrix highlights several relationships between features:

- **Alcohol and Quality** show a moderately positive correlation, indicating that higher alcohol levels may be associated with better quality ratings.
- **Density and Residual Sugar** have a strong positive correlation, suggesting that higher density often indicates higher residual sugar content.
- **Free Sulfur Dioxide and Total Sulfur Dioxide** are strongly positively correlated, as expected, since free sulfur dioxide is a component of the total sulfur dioxide. To avoid redundancy is better to remove one of them. The variable Free.sulfur.dioxide was excluded from further analysis.
- **Volatile Acidity and Quality** appear to have a slight negative correlation, suggesting that higher volatile acidity could be associated with lower wine quality.
- **Density and Residual Sugar**: There is a noticeable positive correlation between density and residual sugar, which suggests that higher sugar content increases the wine's density.

4.3 Correlation Network Graph

To visually explore the relationships between the physicochemical properties in the wine dataset, a correlation network graph was generated. This graph represents variables as nodes and their correlations as edges. Positive correlations are shown with green edges, while negative correlations are represented with red edges, the thickness of each edge reflects the strength of the correlation.

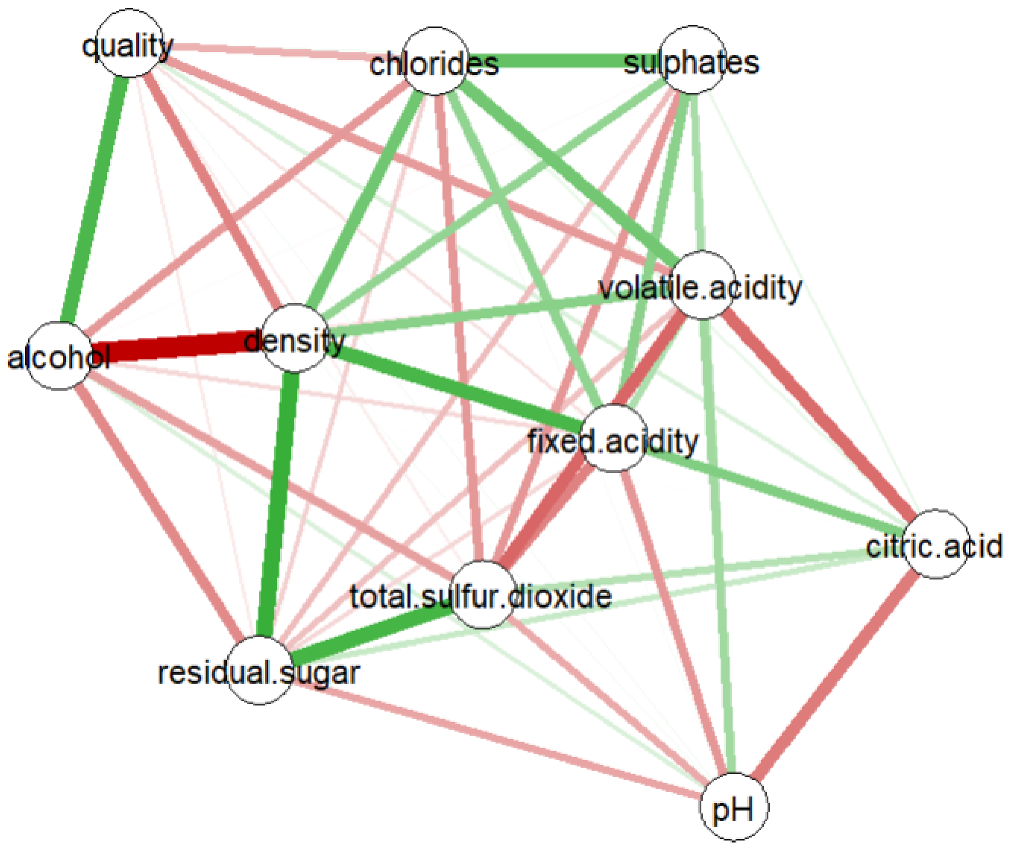


Figure 3: Correlation Network Graph

This graph provides a visual framework for understanding dependencies within the dataset and identifying influential variables.

5 Supervised Learning

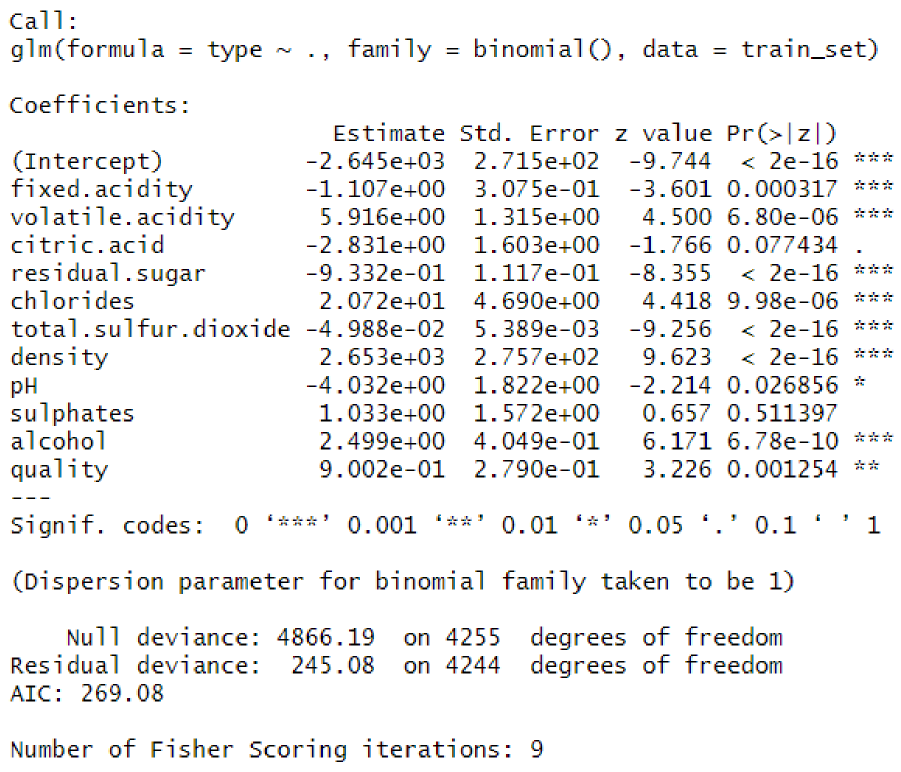
The objective of the supervised learning phase was to classify wine types (red or white) based on their physicochemical properties. Several classification algorithms were implemented:

- Logistic Regression,
- K-Nearest Neighbors (KNN)
- Decision Tree
- Random Forest
- XGBoost

The models were trained and evaluated using an 80/20 train-test split, with cross-validation applied where necessary to optimize model performance. Below, the process and results of each method are detailed.

5.1 Logistic Regression

Logistic regression was applied as a baseline model to predict the wine type. This methodological choice was driven by the logistic regression model's robustness in handling binary outcomes, making it reliable for classification tasks. Logistic regression also provides interpretable coefficients, which facilitate understanding the relationship between physicochemical features and wine type.



```
Call:
glm(formula = type ~ ., family = binomial(), data = train_set)

Coefficients:
                       Estimate Std. Error z value Pr(>|z|)    
(Intercept)          -2.645e+03  2.715e+02  -9.744  < 2e-16 ***
fixed.acidity        -1.107e+00  3.075e-01  -3.601 0.000317 ***
volatile.acidity      5.916e+00  1.315e+00   4.500 6.80e-06 ***
citric.acid          -2.831e+00  1.603e+00  -1.766 0.077434 .  
residual.sugar       -9.332e-01  1.117e-01  -8.355  < 2e-16 ***
chlorides             2.072e+01  4.690e+00   4.418 9.98e-06 ***
total.sulfur.dioxide -4.988e-02  5.389e-03  -9.256  < 2e-16 ***
density               2.653e+03  2.757e+02   9.623  < 2e-16 ***
pH                   -4.032e+00  1.822e+00  -2.214 0.026856 *  
sulphates             1.033e+00  1.572e+00   0.657 0.511397    
alcohol               2.499e+00  4.049e-01   6.171 6.78e-10 ***
quality               9.002e-01  2.790e-01   3.226 0.001254 ** 
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 4866.19  on 4255  degrees of freedom
Residual deviance:  245.08  on 4244  degrees of freedom
AIC: 269.08

Number of Fisher Scoring iterations: 9
```

Figure 4: Logistic Regression

The table provides a detailed overview of how each feature impacts the probability of a wine being classified as red. Significant features with low p-values and high absolute coefficients are strong predictors.

Significant Variables:
Focus on variables with low p-values and significance codes (***, **, *) for predicting wine type.
Variable Effects:

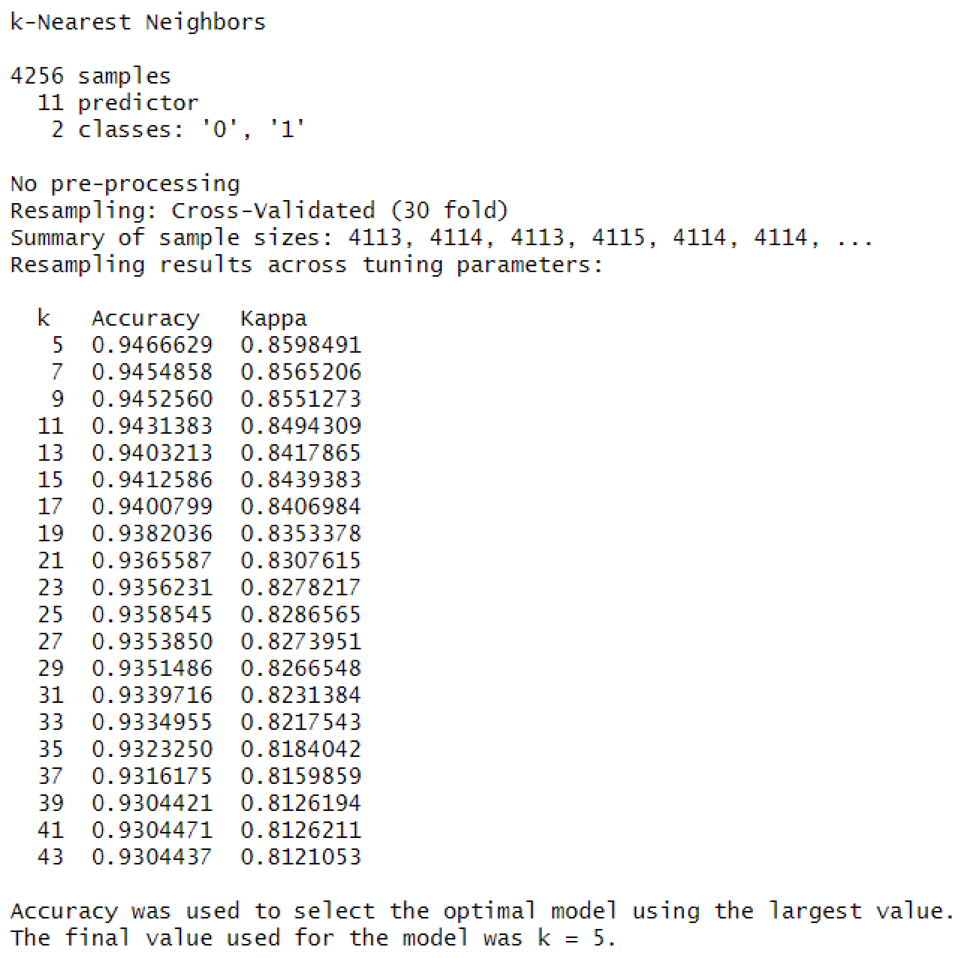
- **Positive Effects**: Volatile acidity, chlorides, free sulfur dioxide, density, alcohol, and quality increase the probability of a wine being red.
- **Negative Effects**: Fixed acidity, residual sugar, total sulfur dioxide, and pH decrease the probability of a wine being red.

Table 2: Logistic Regression Model

Model	**Accuracy**	**Precision**	**Recall**	**F1-score**
Logistic R	0.9891	0.9938	0.9975	0.9957

5.2 K-Nearest Neighbors

Following the logistic regression analysis, the K-Nearest Neighbors (KNN) method was chosen to explore alternative modeling techniques that might provide additional insights in classifying wine types. KNN is a non-parametric approach capable of identifying complex patterns in the data without requiring assumptions about the underlying distribution. This flexibility makes it particularly effective for capturing nonlinear relationships among the physicochemical properties of the wines.



```
k-Nearest Neighbors

4256 samples
  11 predictor
   2 classes: '0', '1'

No pre-processing
Resampling: Cross-Validated (30 fold)
Summary of sample sizes: 4113, 4114, 4113, 4115, 4114, 4114, ...
Resampling results across tuning parameters:

  k   Accuracy   Kappa
   5  0.9466629  0.8598491
   7  0.9454858  0.8565206
   9  0.9452560  0.8551273
  11  0.9431383  0.8494309
  13  0.9403213  0.8417865
  15  0.9412586  0.8439383
  17  0.9400799  0.8406984
  19  0.9382036  0.8353378
  21  0.9365587  0.8307615
  23  0.9356231  0.8278217
  25  0.9358545  0.8286565
  27  0.9353850  0.8273951
  29  0.9351486  0.8266548
  31  0.9339716  0.8231384
  33  0.9334955  0.8217543
  35  0.9323250  0.8184042
  37  0.9316175  0.8159859
  39  0.9304421  0.8126194
  41  0.9304471  0.8126211
  43  0.9304437  0.8121053

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 5.
```

Figure 5: KNN results

Table 3: KNN Model

Model	Accuracy	Precision	Recall	F1-score
KNN	0.9179	0.9572	0.9714	0.9642

5.3 Decision Tree

The next step in our supervised learning analysis involved implementing a Decision Tree model for classifying wine types. Decision Trees offer a unique advantage by generating clear, interpretable decision rules that allow for intuitive understanding of the relationships between the physicochemical properties and wine classification. This simplicity makes Decision Trees particularly appealing for exploratory analysis, as they help identify the most influential features and their thresholds for classification. Moreover, their hierarchical structure provides a visual framework for understanding how the model makes predictions.

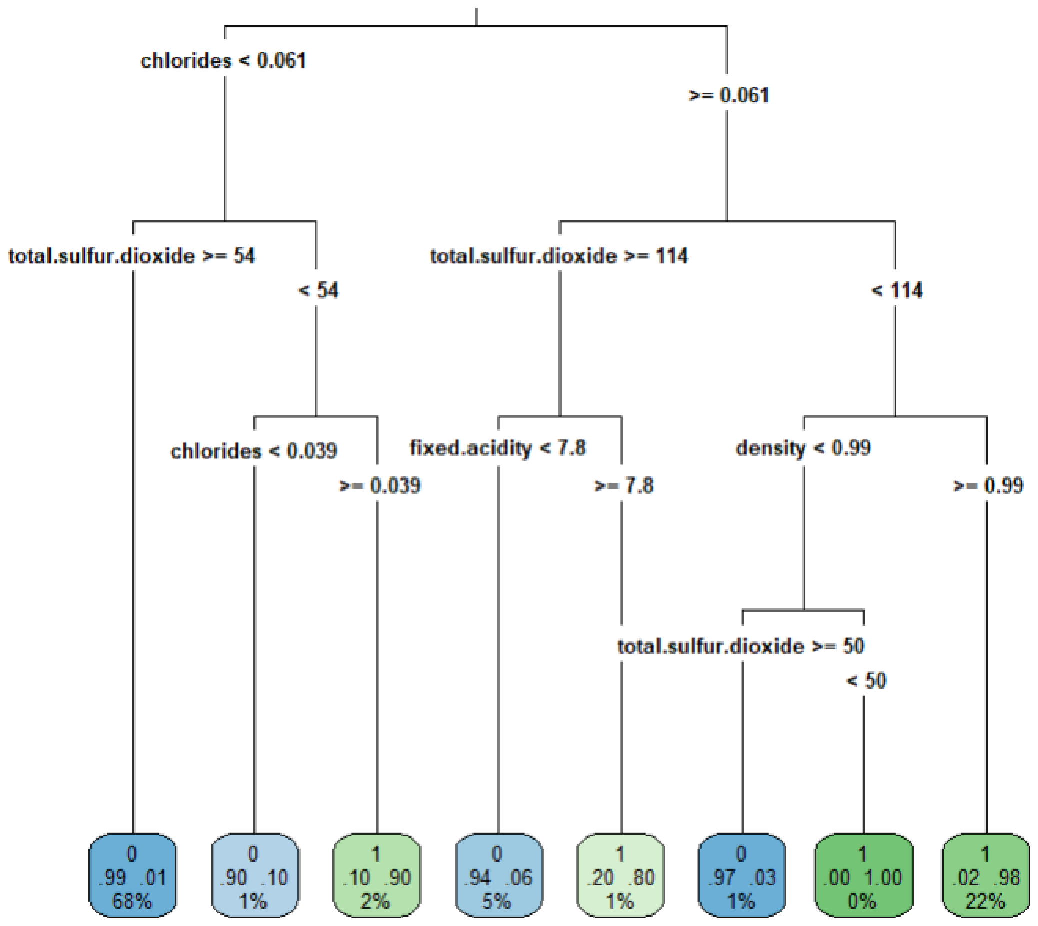


Figure 6: Decision Tree

Table 4: Decision Tree Model

Model	Accuracy	Precision	Recall	F1-score
Decision Tree	0.970	0.985	0.987	0.986

5.4 Random Forest

A Random Forest classifier was used to improve the accuracy of predicting wine types. Key parameters such as *mtry*, *nodesize*, *minsplit*, and *maxdepth* were optimized using a grid search to find the best combination for maximum accuracy. This systematic approach helps the model perform well on new, unseen data.
The Random Forest model underwent hyperparameter tuning to explore the best combination of values for the following parameters:

- **mtry** (Number of Variables Randomly Sampled at Each Split): 3
- **nodesize** (Minimum Number of Samples at Each Terminal Node): 5
- **minsplit** (Minimum Number of Samples Required to Split a Node): 4
- **maxdepth** (Maximum Depth of the Trees): 20

The best combination of parameters was identified using an iterative search, resulting in an impressive accuracy of **99%** on the test set.

Table 5: Random Forest Model

Model	**Accuracy**	**Precision**	**Recall**	**F1-score**
Random For.	0.986	0.991	1	0.995

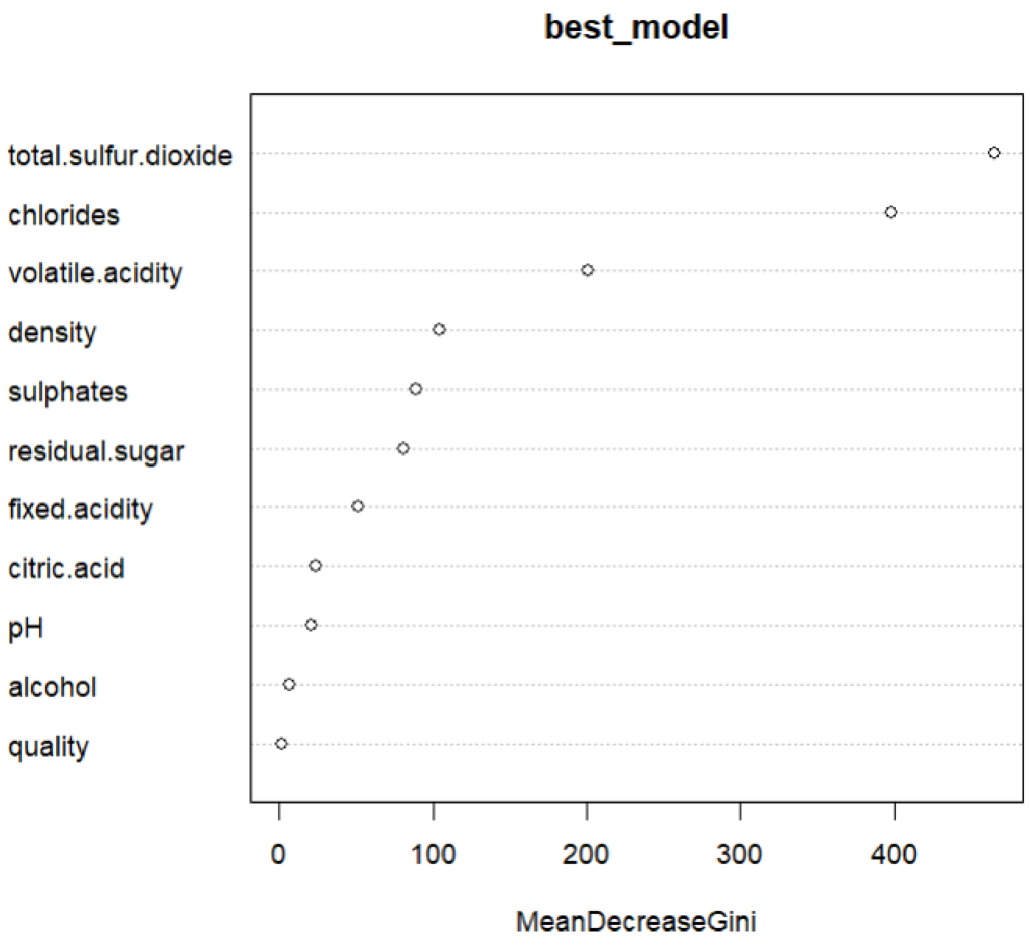


Figure 7: Variables importance plot

5.5 XG Boost

To further enhance the predictive accuracy and robustness of the analysis in classifying wine types, the XGBoost algorithm was integrated into the study. XGBoost is a powerful machine learning technique known for its efficiency and effectiveness across a wide range of classification tasks. Its ability to handle large datasets and reduce overfitting through advanced regularization makes it a standout choice for this type of analysis.

Parallel processing was employed to leverage multiple CPU cores, significantly speeding up the computational workload during the extensive tuning and training processes. This approach was crucial for efficiently handling the large number of hyperparameter combinations tested.

The core of the implementation involved meticulous tuning of the XGBoost model. A comprehensive grid of key hyperparameters was designed, including the number of boosting rounds, tree depth, learning rate, and other parameters critical to the model's performance. This grid search approach enabled the evaluation of numerous parameter combinations to identify the optimal configuration.

To ensure the reliability of the results, a cross-validation strategy with multiple folds was applied. This method tested the model across different subsets of the data, validating its performance and minimizing the risk of overfitting.

After the grid search and cross-validation process, the optimal parameters for the XGBoost model were identified as follows:

- **Number of Boosting Rounds (nrounds):** 100
- **Maximum Depth of Trees (max_depth):** 3
- **Learning Rate (eta):** 0.1
- **Gamma (gamma):** 0
- **Subsample Ratio of Columns (colsample_bytree):** 0.5
- **Minimum Sum of Instance Weight (min_child_weight):** 1
- **Subsample Ratio of Training Instances (subsample):** 1.0

The XGBoost model demonstrated exceptional performance in the comparative analysis of predictive models for classifying wine types. It consistently outperformed other models across all key metrics, including accuracy, precision, recall, and F1 score. The success of XGBoost is attributed to its advanced algorithmic design, which captures complex relationships and interactions within the data while maintaining robustness against overfitting.

These results highlight XGBoost as a highly effective tool for wine classification, offering precise and reliable predictions. Its superior performance underscores its potential in machine learning applications where accuracy and robustness are paramount.

Table 6: XG Boost Model

Model	**Accuracy**	**Precision**	**Recall**	**F1-score**
XG Boost	0.989	0.993	0.998	0.996

6 Results

Table 7: Performance Metrics Supervised Models

Model	**Balanced Accuracy**	**Precision**	**Recall**	**F1 Score**
Logistic Regression	0.989	0.993	0.997	0.995
KNN	0.917	0.957	0.971	0.964
Decision Tree	0.970	0.985	0.987	0.986
Random Forest	0.986	0.991	1	0.995
XGBoost	0.989	0.928	0.722	0.812

This analysis compares several machine learning models to determine which one works best for predicting whether a wine is **red or white**. We used various performance metrics to evaluate how well the models performed, with a key focus on the **F1 score**. The F1 score helps balance **precision** (how many wines labeled as red are actually red) and **recall** (how many red wines were correctly identified). This balance is particularly important for winemakers to avoid mislabeling their wines and maintain high quality standards.

Among all the models tested, **Random Forest** performed the best. It achieved an F1 score of **0.995**, indicating very high accuracy in predicting whether a wine is red or white. The model had a perfect **recall of 1.000**, meaning it correctly labeled every single red or white wine without missing any. Its **precision (0.991)** was also very high, meaning that almost every wine it labeled as red or white was classified correctly. The **balanced accuracy of 0.986** shows that the Random Forest model performs well across both types of wines, making it the best choice for this prediction task.

Logistic Regression also performed very well, achieving an F1 score of **0.995**, with a recall of **0.997** and precision of **0.993**.

The **Decision Tree** model showed solid performance as well, with an F1 score of **0.986**. It had a **precision of 0.985** and a **recall of 0.987**, which means it is fairly reliable at predicting whether a wine is red or white. The **K-Nearest Neighbors (KNN)** model did reasonably well, with an F1 score of **0.964**. It had a **precision of 0.957** and a **recall of 0.971**, suggesting that it can correctly identify red and white wines in most cases, although it still leaves some room for improvement compared to the top models.

On the other hand, **XGBoost** struggled somewhat compared to the others. It had a good **balanced accuracy (0.989)**, but the **recall (0.722)** was relatively low, indicating that it missed a significant number of red or white wines. Its F1 score of **0.812** and **precision of 0.928** suggest that while it is good at correctly labeling wines when it makes a prediction, it has difficulty identifying all true cases.

7 Unsupervised Learning

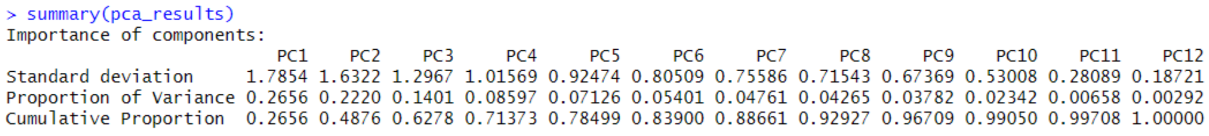
The unsupervised learning part of this study aimed to uncover hidden patterns within the dataset, focusing on distinguishing between red and white wine types based on their physicochemical properties. To do this, advanced methods were used, including Principal Component Analysis (PCA), Uniform Manifold Approximation and Projection (UMAP), and t-Distributed Stochastic Neighbor Embedding (t-SNE). These techniques were combined with K-Means and hierarchical clustering to identify natural groupings and relationships in the dataset.

- **PCA:** Principal Component Analysis (PCA) was used to reduce the dataset's dimensions while keeping most of the important information. This helps make the data simpler and easier to work with.
  - **K-Means:** After PCA, K-Means clustering was applied to group the data into clusters based on similarities. This method is effective for finding central groupings.
  - **Hierarchical Clustering (Ward.D2):** Hierarchical clustering was also used to identify nested structures and relationships within the data. This technique helps reveal how clusters relate to each other in a tree-like structure.
- **UMAP:** Uniform Manifold Approximation and Projection (UMAP) was used to reduce the dataset to two dimensions. This method is great for preserving both local and global patterns in the data, making it useful for visualization.
  - **K-Means:** K-Means clustering was performed on the UMAP-reduced data to identify natural groupings of similar points.
- **t-SNE:** t-Distributed Stochastic Neighbor Embedding (t-SNE) was used to reduce the high-dimensional dataset into two dimensions. t-SNE is particularly good at preserving complex patterns and separating data into meaningful clusters.
  - **K-Means:** K-Means clustering was applied to the t-SNE-reduced data to identify clear and distinct groupings.
  - **Hierarchical Clustering:** Hierarchical clustering was also used on the t-SNE data to explore relationships and uncover additional patterns within the clusters.

Below, the process and results of each method are detailed:

7.1 Principal Component Analysis

Principal Component Analysis (PCA) was applied to the wine dataset to reduce its dimensionality while retaining most of the variance.



```
> summary(pca_results)
Importance of components:
                          PC1    PC2    PC3     PC4     PC5     PC6     PC7     PC8     PC9    PC10    PC11    PC12
Standard deviation     1.7854 1.6322 1.2967 1.01569 0.92474 0.80509 0.75586 0.71543 0.67369 0.53008 0.28089 0.18721
Proportion of Variance 0.2656 0.2220 0.1401 0.08597 0.07126 0.05401 0.04761 0.04265 0.03782 0.02342 0.00658 0.00292
Cumulative Proportion  0.2656 0.4876 0.6278 0.71373 0.78499 0.83900 0.88661 0.92927 0.96709 0.99050 0.99708 1.00000
```

Figure 8: PCA Importance of Components

The biplot visualizes the distribution of wines (red and white) based on the first two principal components, which together explain **48.8%** of the total variance in the dataset. In the plot, the red circles represent one type of wine (white), while the blue triangles represent another type (red).

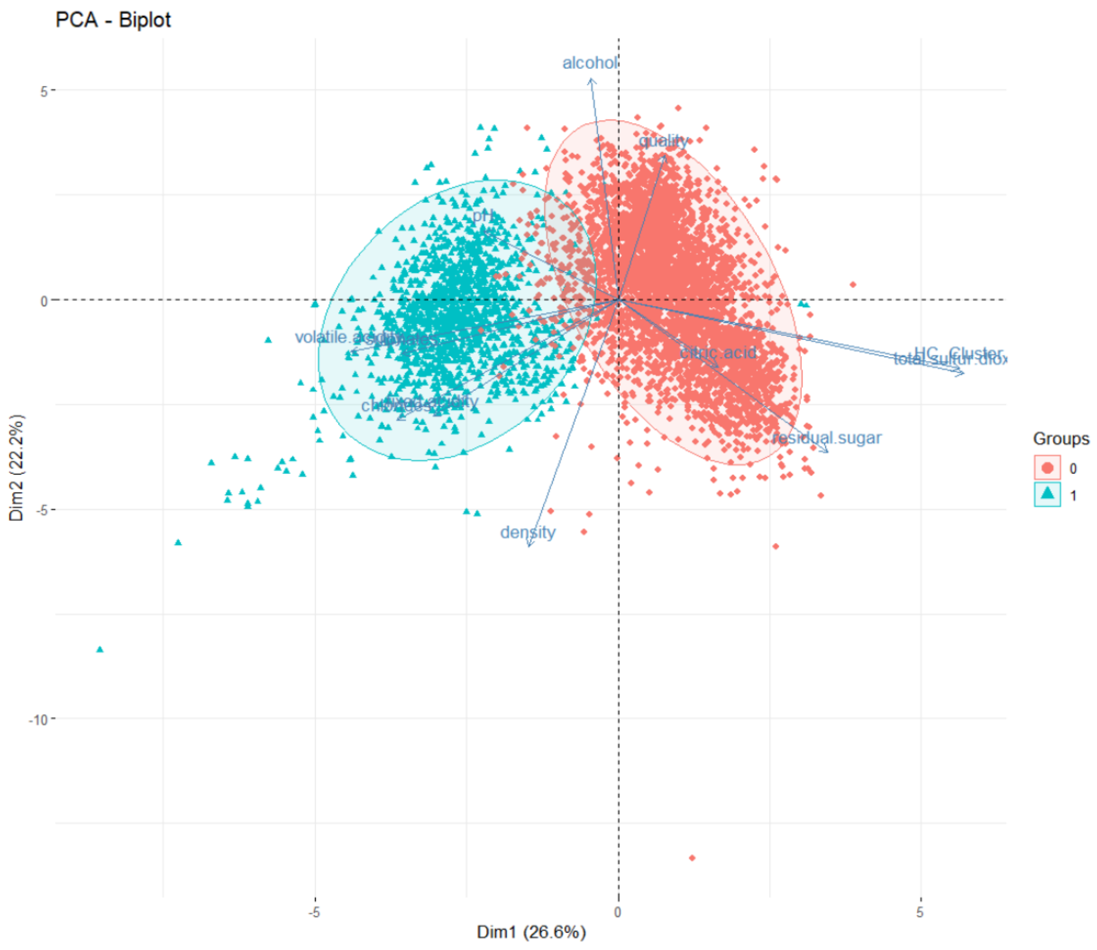


Figure 9: PCA Biplot

The arrows in the biplot indicate the contributions of each variable to the principal components. Longer arrows signify a stronger influence on the direction of the components. For example:

- **Residual sugar** and **total sulfur dioxide** show a strong positive contribution to the right-hand cluster (likely corresponding to white wines).
- **Fixed acidity**, **volatile acidity**, and **chlorides** are more associated with the left-hand cluster (likely corresponding to red wines).

The table summarizing the importance of components highlights the following key points:

- **PC1** explains **26.6%** of the variance, and **PC2** explains **22.2%**. Together, they account for nearly **48.8%** of the data's variability.
- Beyond **PC4**, the variance contribution of subsequent components diminishes significantly, suggesting that focusing on the first few components is sufficient for capturing the main patterns in the data. The following analysis were computed based on the first 5 principal components, which captured the **78.5%** of the explained variance.

7.2 PCA with K-means clustering

Principal Component Analysis (PCA) was applied to reduce the dimensionality of the dataset while retaining its essential structure. This allowed us to visualize and analyze the data in a lower-dimensional space. Following the PCA, K-Means clustering was implemented to identify natural groupings within the data.

The **Elbow Method** was used to determine the optimal number of clusters. This method suggests that the optimal cluster count is where the decrease in the total within-cluster variance slows down significantly, forming an "elbow" shape in the graph. The elbow was observed around two to four clusters.

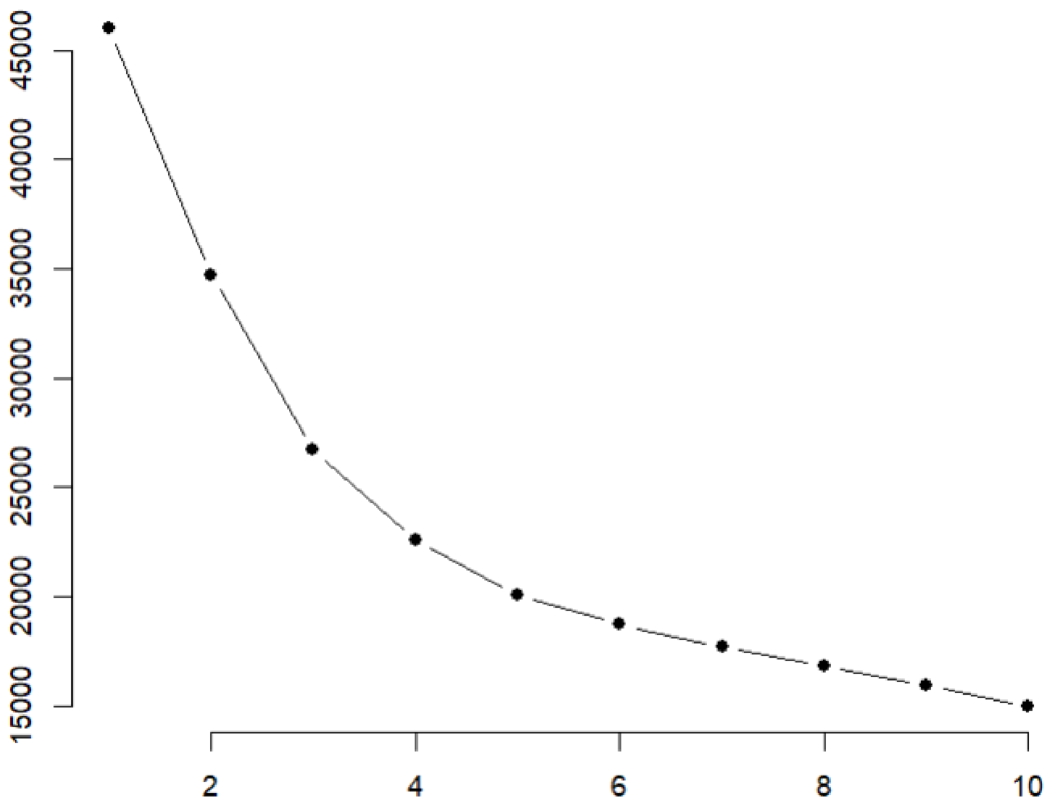


Figure 10: Elbow Method

The Elbow Method was used to determine the optimal number of clusters. This method suggests that the optimal cluster count is where the decrease in the total within-cluster variance slows down significantly, forming an "elbow" shape in the graph. The elbow was observed around two to four clusters.

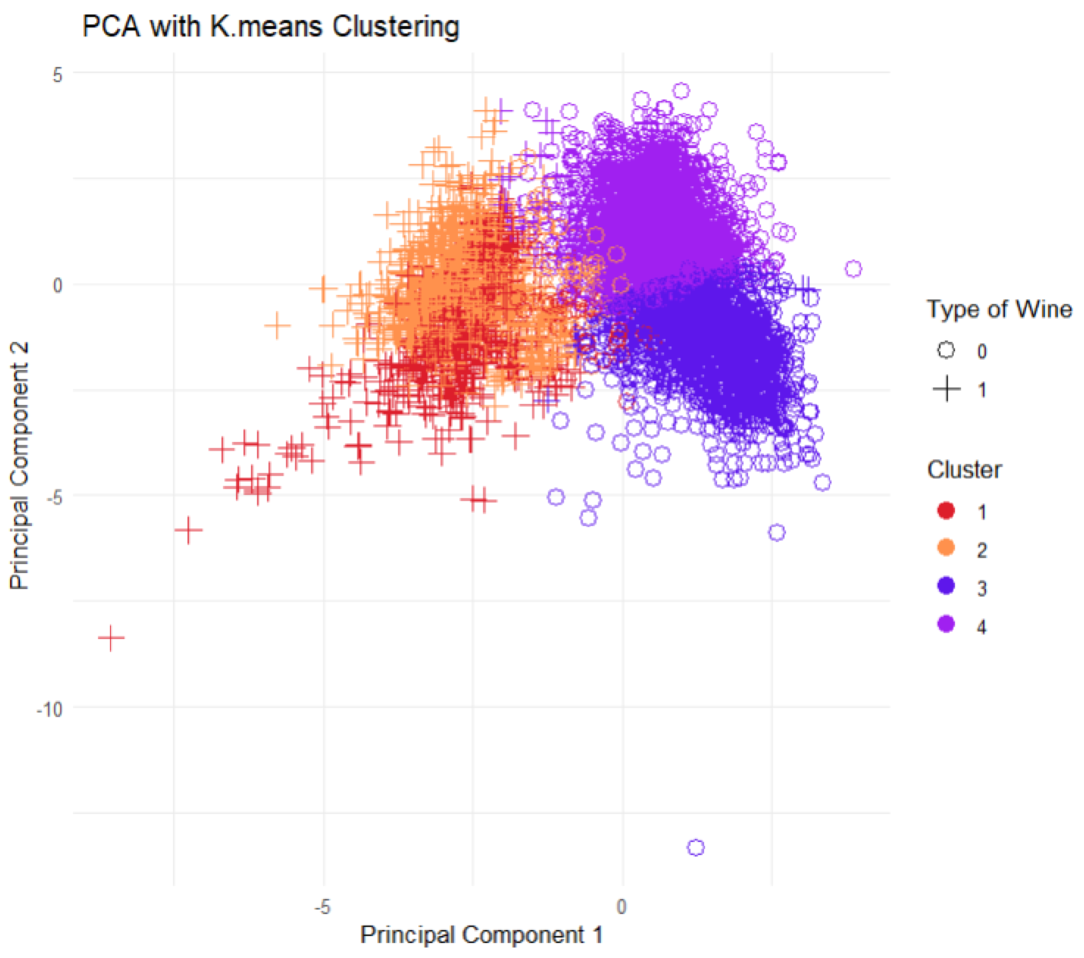


Figure 11: K-means Clustering (4 Clusters)

In the 4-cluster configuration, the data is separated into finer subgroups, with clusters showing some overlap, but each grouping captures unique properties of the wines.

Table 8: Silhouette Scores for 4 Clusters

Cluster	**Size**	**Average Silhouette Width**
1	879	0.34
2	2232	0.31
3	1665	0.30
4	544	0.21

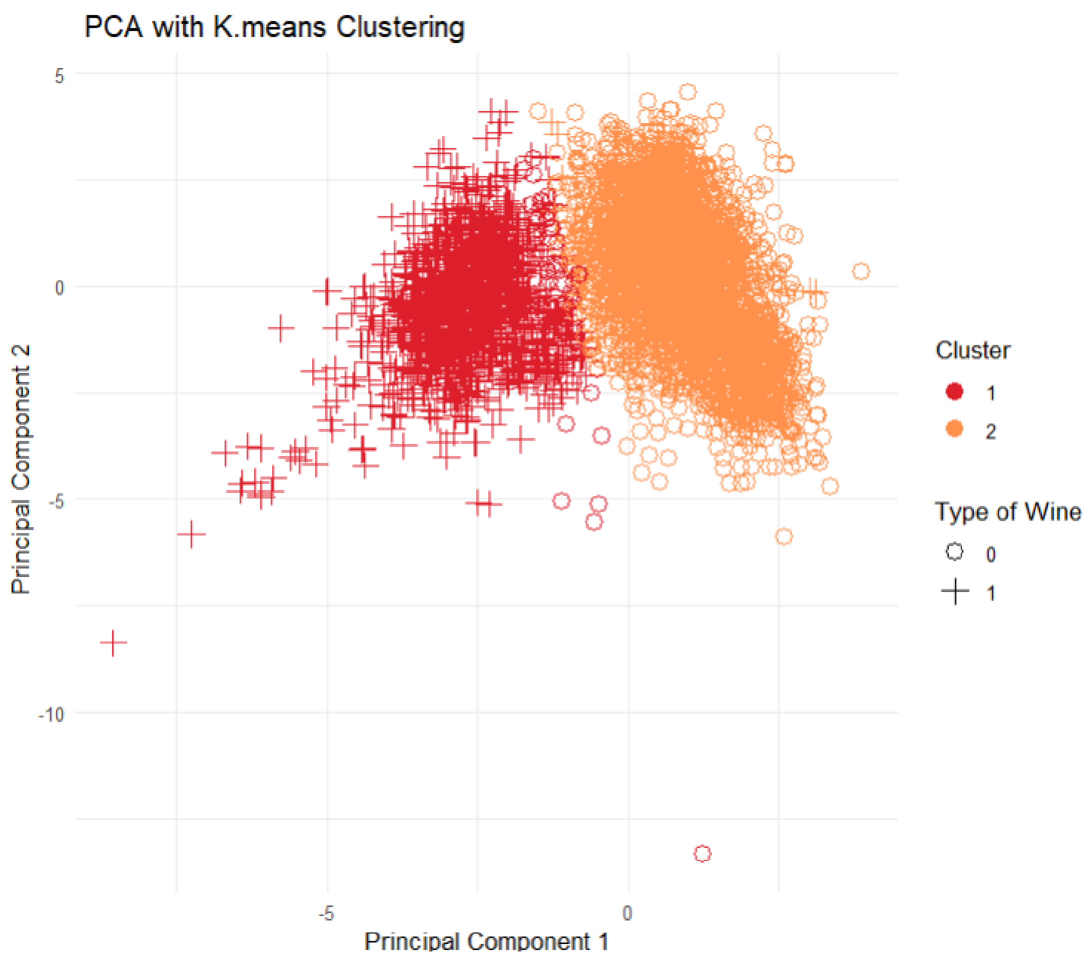


Figure 12: K-means Clustering (2 Clusters)

The 2-cluster configuration clearly separates red wines (Cluster 1) and white wines (Cluster 2), aligning with the type of wine, as indicated in the plot legends.

Table 9: Silhouette Scores for 2 Clusters

Cluster	**Size**	**Average Silhouette Width**
1	3929	0.34
2	1391	0.22

This suggests that the 2-cluster configuration may better capture the natural grouping of the data, specifically the division between red and white wines.

7.3 PCA Hierarchical Clustering Ward.D2

In addition to K-Means clustering, hierarchical clustering was applied to the PCA-reduced data to explore nested relationships and provide further structural insights into the data. The Ward.D2 method, which minimizes the variance within clusters, was used to determine cluster groupings.

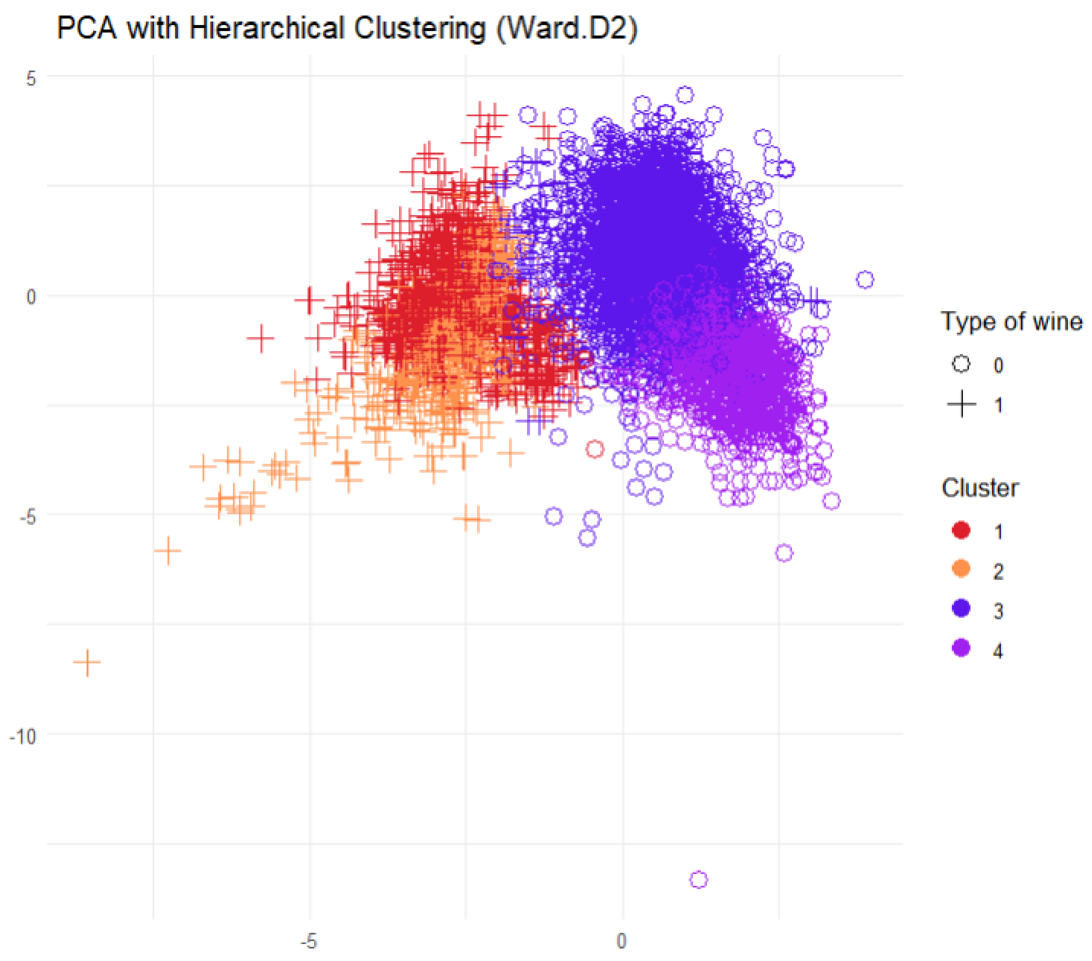


Figure 13: Hierarchical Clustering Ward.D2)

The plot highlights four distinct clusters that align closely with the two main wine types (red and white). The clusters illustrate a clear separation between red and white wine types, with some overlap reflecting the shared physicochemical characteristics between certain samples.

- Clusters 1 and 2 primarily represent red wine samples, with finer subdivisions observed based on physicochemical properties.
- Clusters 3 and 4 mostly include white wine samples, with variability in sugar content and acidity influencing their separation.

Table 10: Silhouette Analysis for Hierarchical Clustering (Ward.D2)

Cluster	**Size**	**Average Silhouette Width**
1	886	0.33
2	521	0.21
3	2809	0.23
4	1104	0.40
Overall	—	**0.28**

7.4 UMAP

These visualizations highlight how the data points are grouped into distinct clusters, with each cluster color-coded. The clustering was performed using both KMeans and Hierarchical methods.

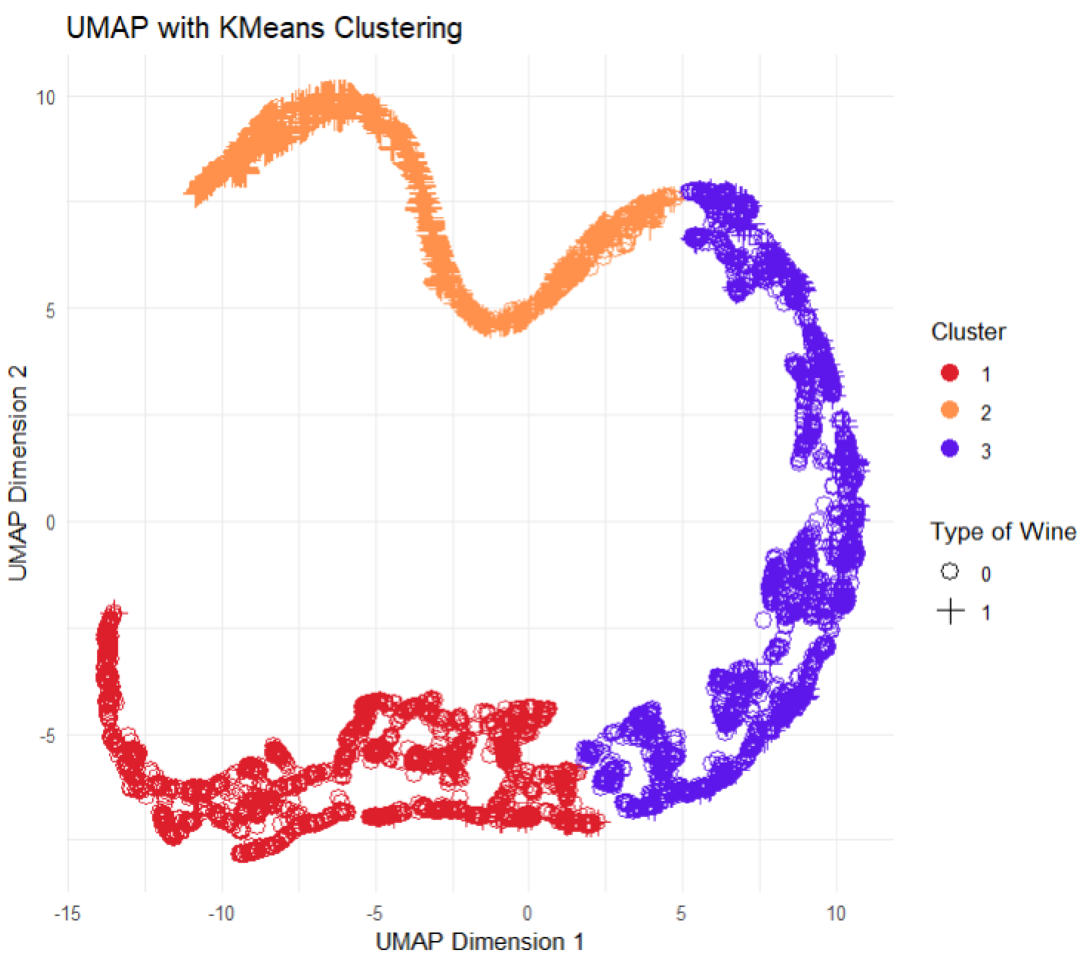


Figure 14: UMAP k-means

- The UMAP projection enabled distinct cluster separations, resulting in three clusters.
- Clusters 1 and 2 showed compact and well-defined groupings, while Cluster 3 exhibited slightly overlapping boundaries with Cluster 2.
- The silhouette analysis for KMeans revealed an average silhouette width of 0.53, with Cluster 2 having the highest score (0.55).

The silhouette analysis was conducted to evaluate the clustering performance of UMAP-reduced data using both KMeans and Hierarchical clustering methods. Below is the summary of the results:

Table 11: Silhouette Analysis for UMAP Clustering

Clustering Method	**Cluster**	**Size**	**Average Silhouette Width**
KMeans	1	1755	0.54
	2	1472	0.55
	3	2093	0.49
	Average	–	**0.53**

7.5 T-SNE

As the exploration of clustering within the dataset advances, focus has shifted to t-Distributed Stochastic Neighbor Embedding (t-SNE), a dimensionality reduction technique that excels in uncovering non-linear relationships within high-dimensional data. The objective of applying t-SNE is to investigate whether it can provide better-defined clusters compared to earlier methods like PCA and UMAP.

t-SNE with K-Means Clustering

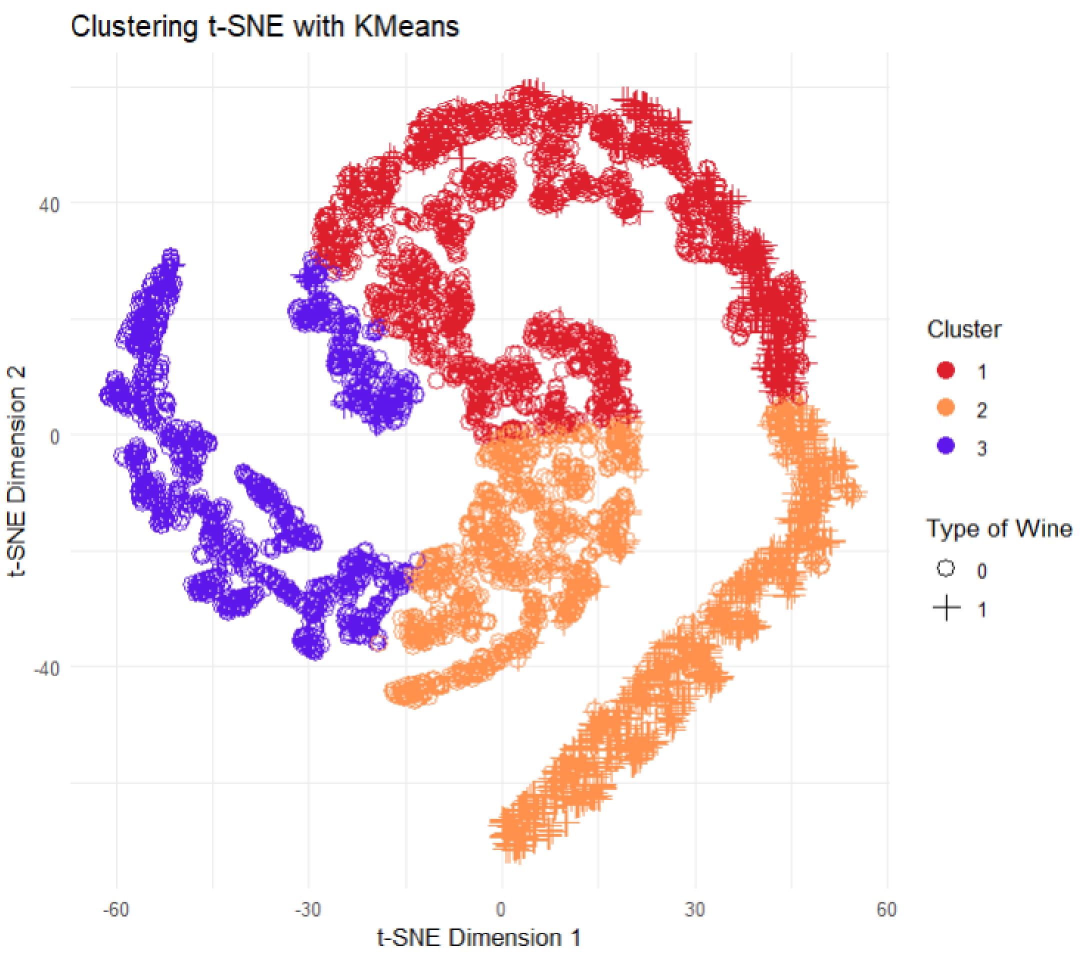
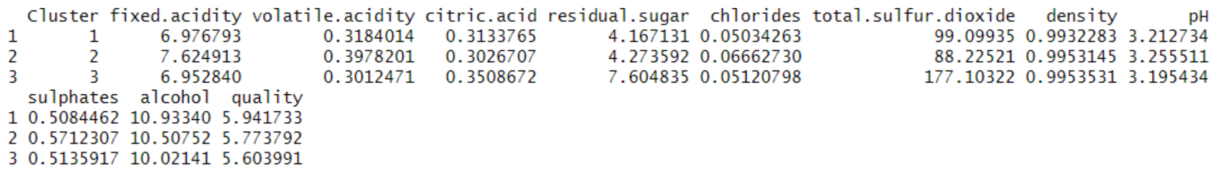


Figure 15: T-SNE with k-means



```
  Cluster fixed.acidity volatile.acidity citric.acid residual.sugar  chlorides total.sulfur.dioxide   density       pH
1       1      6.976793        0.3184014   0.3133765       4.167131 0.05034263             99.09935 0.9932283 3.212734
2       2      7.624913        0.3978201   0.3026707       4.273592 0.06662730             88.22521 0.9953145 3.255511
3       3      6.952840        0.3012471   0.3508672       7.604835 0.05120798            177.10322 0.9953531 3.195434
  sulphates  alcohol  quality
1 0.5084462 10.93340 5.941733
2 0.5712307 10.50752 5.773792
3 0.5135917 10.02141 5.603991
```

Figure 16: Average Values for Physicochemical Properties within Clusters

- **Cluster 1: Balanced and High-Quality Wines**
  This cluster represents wines that are well-balanced and exhibit the highest quality among all clusters. The wines in this group are characterized by moderate levels of **fixed acidity** and **volatile acidity**, providing a smooth and harmonious flavor profile. The **residual sugar** is low, consistent with dry wines, while the **alcohol content** is the highest. The wines in this cluster also show an average **quality score** of 5.94, indicating high consumer appeal.

- **Cluster 2: Sharp and Average-Quality Wines**
  Wines in this cluster are characterized by high **fixed acidity** and **volatile acidity**. These wines have low **residual sugar** and moderate **alcohol content**. With the lowest **total sulfur dioxide**, these wines prioritize a more natural preservation approach. The cluster's **quality score** is slightly lower at 5.77, suggesting that the wines appeal less to consumers.

- **Cluster 3: Sweet and Lower-Quality Wines**
  This cluster is defined by its sweeter flavor profile, featuring the highest **residual sugar** levels and the lowest **alcohol content**. These wines are also characterized by the highest **total sulfur dioxide**. The average **quality score** (5.60) is the lowest among the clusters. This indicates that these wines appeal a niche market.

.

t-SNE with Hierarchical Clustering

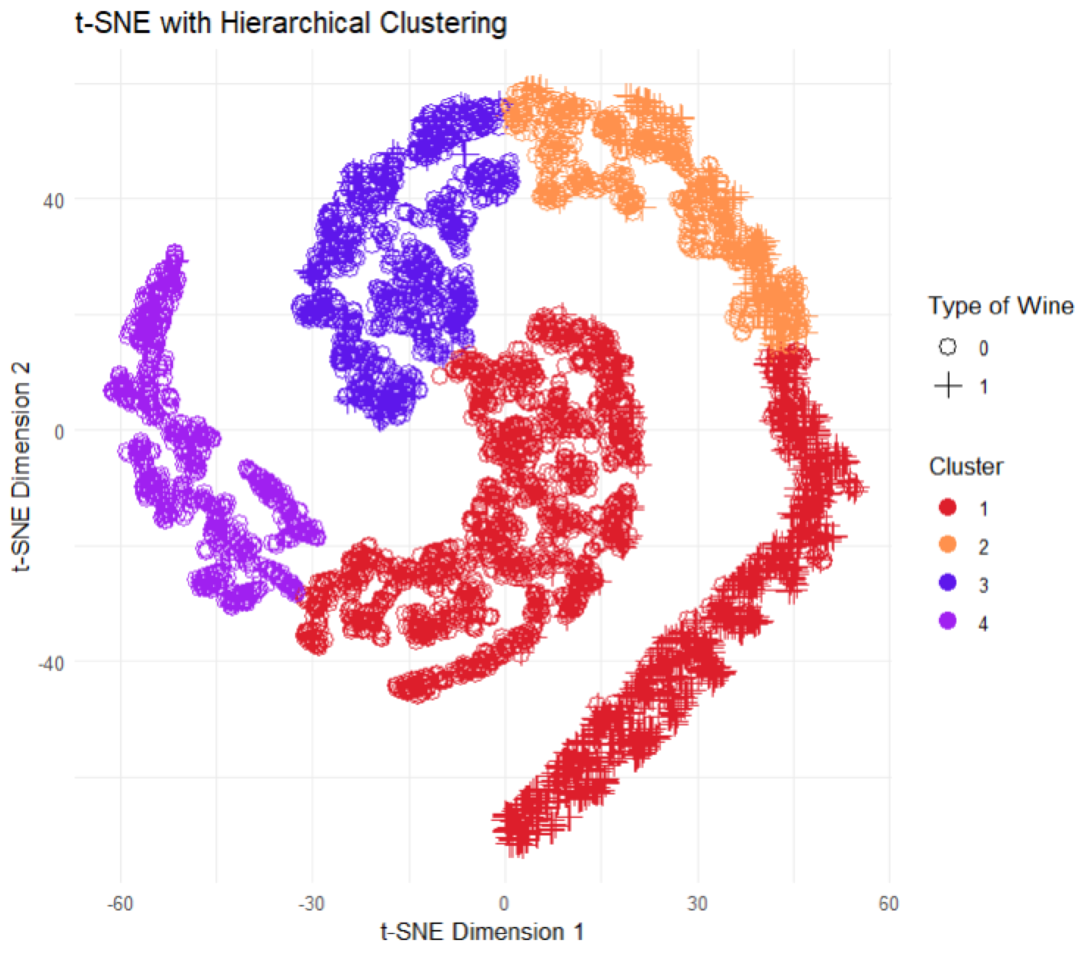


Figure 17: T-SNE with Hierarchical Clustering

- **Cluster 1 :** This cluster dominates the lower right area of the plot and includes wines of both types.
- **Cluster 2 :** Occupying the middle region, this cluster overlaps with both red and white wine types, indicating a mix of properties shared between the two.
- **Cluster 3 :** Located on the top-left of the plot, this cluster is distinctly separated from others and likely represents a subset of wine samples dominated by white wine.
- **Cluster 4 :** This cluster includes primarily wines of one type (white). It may include outliers or samples with unique characteristics.

Silhouette Analysis To assess the clustering quality, silhouette scores were computed. The table below summarizes the average silhouette widths for both clustering methods:

Table 12: Silhouette Scores for t-SNE Clustering Methods

Clustering Method	Cluster	Average Silhouette Width
t-SNE + K-Means	Cluster 1	0.38
	Cluster 2	0.38
	Cluster 3	0.44
Overall Average		0.40
t-SNE + Hierarchical	Cluster 1	0.23
	Cluster 2	0.50
	Cluster 3	0.46
	Cluster 4	0.54
Overall Average		0.36

K-Means clustering demonstrated a slightly higher overall silhouette width, with an average of 0.40, compared to 0.36 for hierarchical clustering. However, hierarchical clustering identified a higher silhouette score for specific clusters (e.g., Cluster 4 with 0.54), suggesting it may better capture certain group structures. These findings highlight the complementary strengths of the two approaches.

8 Results

This analysis evaluated clustering methods to effectively distinguish between **red and white wines** and explore deeper relationships between variables. **Principal Component Analysis (PCA)** was applied initially to reduce data dimensionality, which allowed clearer visualization and clustering. Following PCA, **K-Means clustering** indicated that the **2-cluster configuration** provided the best separation, aligning well with red and white wine categories.

Hierarchical Clustering (Ward.D2) explored nested relationships, identifying four clusters, two primarily red and two primarily white,within the clusters finer distinctions based on acidity and sugar content were made. **UMAP**, combined with K-Means, produced well-defined clusters with an **average silhouette score of 0.53**, indicating strong separation for certain subgroups.

t-SNE was then used to uncover non-linear relationships, identifying three main clusters representing different wine qualities, such as balanced high-quality wines versus sweeter, lower-quality wines.

The **2-cluster K-Means** (PCA-reduced) configuration stood out as the simplest and most effective model for distinguishing between red and white wines.

In conclusion, the **2-cluster K-Means** (PCA-reduced) configuration stood out as the simplest and most effective model for distinguishing between red and white wines, while **K-Means with PCA** provided straightforward and accurate classification, **UMAP** and **t-SNE** offered additional insights by highlighting more detailed wine properties.

9 Key Findings

Prediction Models Performance

- **Random Forest** emerged as the most effective model for predicting whether a wine is **red or white**, demonstrating high **accuracy** with an **F1 score of 0.995**, perfect **recall (1.000)**, and exceptional **precision (0.991)**. This makes Random Forest the top choice for ensuring accurate wine classification with minimal errors.
- **Logistic Regression** also showed strong performance, closely matching Random Forest with an **F1 score of 0.995**, **recall of 0.997**, and **precision of 0.993**, offering a reliable alternative with only a slight reduction in recall.

Clustering Analysis

- **PCA with K-Means clustering** effectively reduced dimensionality, making visualization clearer and revealing that a **2-cluster configuration** aligned well with red and white wine distinctions. This straightforward approach allowed easy classification with minimal overlap.
- **Hierarchical Clustering (Ward.D2)** provided deeper insights, identifying **four clusters**, split into two red and two white categories, which highlighted finer distinctions influenced by properties such as **acidity** and **sugar content**.

Advanced Techniques

- The application of **UMAP** and **t-SNE** provided a more nuanced understanding of wine characteristics:
  - **UMAP**, combined with K-Means, formed distinct and well-defined clusters with an **average silhouette score of 0.53**, effectively showing separation based on wine type.
  - **t-SNE** uncovered complex relationships, distinguishing **three main clusters** representing different wine qualities—balanced high-quality wines, sharper average-quality wines, and sweeter lower-quality wines. These analyses offer valuable insights into the diversity of wine profiles beyond simple red and white categorization.

Conclusion

Random Forest proved to be the most reliable model for accurately classifying wine type, while **K-Means clustering with PCA** served as an effective method for straightforward separation between red and white wines. The use of **UMAP** and **t-SNE** further explored unique wine properties.

10 R Code

```r
library(corrplot)
library(ggplot2)
library(reshape2)
library(caret)
library(rpart)
library(rpart.plot)
library(randomForest)
library(xgboost)
library(parallel)
library(doParallel)
library(factoextra)
library(cluster)
library(umap)
library(Rtsne)
library(qgraph)

# Load red and white wine datasets
red_wine <- read.csv("C:/Users/melan/OneDrive/Documenti/winequality-red.csv", sep = ";")
white_wine <- read.csv("C:/Users/melan/OneDrive/Documenti/winequality-white.csv", sep = ";")

# Add wine type to each dataset
red_wine$type <- "red"
white_wine$type <- "white"

# Combine datasets
combined_wine <- rbind(red_wine, white_wine)

head(combined_wine)
summary(combined_wine)

# Ensure column names match
names(white_wine) <- names(red_wine)

sum(is.na(combined_wine))

combined_wine <- unique(combined_wine)

# Remove the 'free.sulfur.dioxide' variable from the dataset
combined_wine <- combined_wine[, !(names(combined_wine) %in% "free.sulfur.dioxide")]

# Check the number of rows before and after removing duplicates
original_rows <- nrow(combined_wine)
unique_rows <- nrow(combined_wine)
cat("Number of duplicate rows removed:", original_rows - unique_rows, "\n")

#Type distribution
type_distribution <- table(combined_wine$type)
pie(type_distribution,
    labels = paste(names(type_distribution),
                   "(", round(100 * type_distribution / sum(type_distribution), 1), "%)", sep = ""),
    main = "Distribuzione dei tipi di vino",
    col = c("#c45bad", "#efd79d"))


# Select only numeric columns
numeric_data <- combined_wine[sapply(combined_wine, is.numeric)]

# Function to normalize values between 0 and 1
normalize <- function(x) {
  return((x - min(x)) / (max(x) - min(x)))
}

# Apply normalization to all numeric columns
normalized_data <- as.data.frame(lapply(combined_wine, function(col) {
  if (is.numeric(col)) {
    normalize(col)
  } else {
    col  # Keep non-numeric columns unchanged
  }
}))

# Check the normalized dataset
summary(normalized_data)
```

```r
# Calculate the correlation matrix
correlation_matrix <- cor(numeric_data, use = "complete.obs")

# Display the correlation matrix
print(correlation_matrix)

# Check the structure of the dataset
str(combined_wine)

# Convert 'type' to binary numeric
combined_wine$type <- ifelse(combined_wine$type == "red", 1, 0)

# Verify the conversion
table(combined_wine$type)

str(combined_wine)

# Verify that all numeric columns are between 0 and 1
sapply(normalized_data, function(col) {
  if (is.numeric(col)) {
    range(col)  # Check the range of each numeric column
  }
})

# Transform the correlation matrix into long format
correlation_data <- melt(correlation_matrix)

# Create the heatmap with ggplot2
ggplot(correlation_data, aes(Var1, Var2, fill = value)) +
  geom_tile(color = "white") +
  scale_fill_gradient2(low = "#d8849e", high = "#c45bad", mid = "white", midpoint = 0,
                       limit = c(-1, 1), space = "Lab") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, vjust = 1, hjust = 1)) +
  labs(title = "Heatmap della Matrice di Correlazione", x = "", y = "")

# Remove the 'free.sulfur.dioxide' variable from the dataset
combined_wine <- combined_wine[, !(names(combined_wine) %in% "free.sulfur.dioxide")]

# Check the structure of the updated dataset
str(combined_wine)

# Create the correlation network graph
qgraph(correlation_matrix,
       layout = "spring",
       vsize = 6,
       label.cex = 1.2,
       label.scale = FALSE,
       labels = colnames(correlation_matrix),
       label.norm = "0",
       title = "Correlation Network Graph",
       label.color = "black")

# SUPERVISED LEARNING

# Split the dataset into training and test sets
set.seed(123)  # For reproducibility
train_index <- createDataPartition(combined_wine$type, p = 0.8, list = FALSE)
train_set <- combined_wine[train_index, ]
test_set <- combined_wine[-train_index, ]

# Logistic regression model
model <- glm(type ~ ., data = train_set, family = binomial())

# Model summary
summary(model)

# Predictions on the test set
predictions <- predict(model, test_set, type = "response")

# Convert probabilities to classes
predicted_classes <- ifelse(predictions > 0.5, 1, 0)

# Calculate the confusion matrix
confusionMatrixData <- confusionMatrix(as.factor(predicted_classes), as.factor(test_set$type))
```

```r
# Print the results
print(confusionMatrixData$table)  # Confusion matrix
print(confusionMatrixData$overall['Accuracy'])  # Accuracy
print(confusionMatrixData$byClass['Balanced Accuracy'])  # Balanced Accuracy
print(confusionMatrixData$byClass['Precision'])  # Precision
print(confusionMatrixData$byClass['Recall'])  # Recall
print(confusionMatrixData$byClass['F1'])  # F1-score

# Ensure 'type' is a factor for KNN
train_set$type <- as.factor(train_set$type)
test_set$type <- as.factor(test_set$type)

# Set up training control with cross-validation
train_control <- trainControl(method = "cv", number = 30, savePredictions = "final")

# Set seed for reproducibility
set.seed(111)

# Train KNN model with parameter tuning
knn_model <- train(
  type ~ .,                 # Formula (target variable and predictors)
  data = train_set,         # Training set
  method = "knn",           # KNN method
  trControl = train_control, # Training control
  tuneLength = 20           # Number of k values to test
)

# Print model details to see the best k value
print(knn_model)

# Predictions on the test set
knn_predictions <- predict(knn_model, newdata = test_set)

# Evaluate model performance on the test set
knn_results <- confusionMatrix(knn_predictions, test_set$type)

# Print confusion matrix
print(knn_results$table)

# Evaluation metrics
cat("Balanced Accuracy:", knn_results$byClass['Balanced Accuracy'], "\n")
cat("Precision:", knn_results$byClass['Precision'], "\n")
cat("Recall:", knn_results$byClass['Recall'], "\n")
cat("F1 Score:", knn_results$byClass['F1'], "\n")

# Ensure 'type' is a factor
train_set$type <- as.factor(train_set$type)
test_set$type <- as.factor(test_set$type)

# Create Decision Tree model
tree_model <- rpart(type ~ ., data = train_set, method = "class")

# Model summary
summary(tree_model)

# Visualize the decision tree
rpart.plot(tree_model, type = 3, extra = 104, fallen.leaves = TRUE, cex = 0.7)

# Predictions on the test set
predictions <- predict(tree_model, newdata = test_set, type = "class")

# Evaluate the model
confusionMatrixData <- confusionMatrix(as.factor(predictions), as.factor(test_set$type))

# Print confusion matrix
print(confusionMatrixData$table)

# Evaluation metrics
print(paste("Accuracy:", confusionMatrixData$overall['Accuracy']))
print(paste("Balanced Accuracy:", confusionMatrixData$byClass['Balanced Accuracy']))
print(paste("Precision:", confusionMatrixData$byClass['Precision']))
print(paste("Recall:", confusionMatrixData$byClass['Recall']))
print(paste("F1 Score:", confusionMatrixData$byClass['F1']))

# Ensure 'type' is a factor
train_set$type <- as.factor(train_set$type)
```

```r
test_set$type <- as.factor(test_set$type)

set.seed(13)

# Define parameter values to optimize
mtry_values <- seq(2, sqrt(ncol(train_set) - 1), by = 1)
nodesize_values <- c(1, 5, 10, 20)
minsplit_values <- c(2, 4, 6, 10)
maxdepth_values <- c(5, 10, 15, 20)

# Initialize variables to track the best model
best_model <- NULL
best_accuracy <- 0
best_params <- list()

# Iterate over all combinations of parameters
for (mtry in mtry_values) {
  for (nodesize in nodesize_values) {
    for (minsplit in minsplit_values) {
      for (maxdepth in maxdepth_values) {
        # Train Random Forest model
        model <- randomForest(
          type ~ .,
          data = train_set,
          mtry = mtry,
          nodesize = nodesize,
          ntree = 500,
          maxnodes = maxdepth
        )

        # Predictions on the test set
        predictions <- predict(model, newdata = test_set)

        # Calculate accuracy
        cm <- confusionMatrix(predictions, test_set$type)
        accuracy <- cm$overall['Accuracy']

        # Update best model if accuracy is higher
        if (accuracy > best_accuracy) {
          best_model <- model
          best_accuracy <- accuracy
          best_params <- list(
            mtry = mtry,
            nodesize = nodesize,
            minsplit = minsplit,
            maxdepth = maxdepth
          )
        }
      }
    }
  }
}

# Print best parameters and accuracy
cat("Best Parameters:\n")
print(best_params)
cat("Best Accuracy:", best_accuracy, "\n")

# Evaluate the best model
best_predictions <- predict(best_model, newdata = test_set)
best_cm <- confusionMatrix(best_predictions, test_set$type)

# Print confusion matrix and metrics
print(best_cm$table)
cat("Accuracy:", best_cm$overall['Accuracy'], "\n")
cat("Balanced Accuracy:", best_cm$byClass['Balanced Accuracy'], "\n")
cat("Precision:", best_cm$byClass['Precision'], "\n")
cat("Recall:", best_cm$byClass['Recall'], "\n")
cat("F1 Score:", best_cm$byClass['F1'], "\n")

# Variable importance plot
varImpPlot(best_model)

# Set up parallel processing
num_cores <- detectCores() - 1
cl <- makeCluster(num_cores)
```

```r
registerDoParallel(cl)

# Prepare data for XGBoost
train_set$type <- as.factor(as.numeric(as.factor(train_set$type)) - 1)
test_set$type <- as.factor(as.numeric(as.factor(test_set$type)) - 1)

train_matrix <- as.matrix(train_set[, -which(names(train_set) == "type")])
train_label <- train_set$type

test_matrix <- as.matrix(test_set[, -which(names(test_set) == "type")])
test_label <- test_set$type

# Training control with cross-validation
train_control <- trainControl(
  method = "cv",
  number = 5,
  verboseIter = TRUE,
  allowParallel = TRUE
)

# Tuning grid
tune_grid <- expand.grid(
  nrounds = c(50, 100),          # Number of iterations
  max_depth = c(3, 6, 9, 12),    # Maximum depth
  eta = c(0.01, 0.05, 0.1, 0.3), # Learning rate
  gamma = c(0, 0.1, 0.5, 1),     # Minimum loss reduction
  colsample_bytree = c(0.5, 0.7, 0.9, 1.0), # Column subsampling
  min_child_weight = c(1, 5, 10), # Minimum child weight
  subsample = c(0.5, 0.7, 0.9, 1.0) # Row subsampling
)

# Train XGBoost model
set.seed(111)
xgb_tuned_model <- train(
  x = train_matrix,
  y = train_label,
  method = "xgbTree",
  trControl = train_control,
  tuneGrid = tune_grid
)

# Stop parallel processing
stopCluster(cl)
registerDoSEQ()

# Print best tuning parameters
cat("Best Tuning Parameters:\n")
print(xgb_tuned_model$bestTune)

# Predictions with the optimized model
best_predictions <- predict(xgb_tuned_model, newdata = test_matrix)

# Align levels of predictions and test labels
best_predictions <- factor(best_predictions, levels = levels(test_label))
test_label <- factor(test_label, levels = levels(best_predictions))

# Generate confusion matrix
confusionMatrixData <- confusionMatrix(best_predictions, test_label)

# Print confusion matrix
cat("\nConfusion Matrix:\n")
print(confusionMatrixData$table)

# Print performance metrics
cat("\nPerformance Metrics:\n")
cat("Accuracy:", confusionMatrixData$overall['Accuracy'], "\n")
cat("Balanced Accuracy:", confusionMatrixData$byClass['Balanced Accuracy'], "\n")
cat("Precision:", confusionMatrixData$byClass['Pos Pred Value'], "\n")
cat("Recall:", confusionMatrixData$byClass['Sensitivity'], "\n")
cat("F1 Score:", confusionMatrixData$byClass['F1'], "\n")

# UNSUPERVISED LEARNING

# Prepare data for PCA
pca_data <- combined_wine[, sapply(combined_wine, is.numeric)]
pca_data <- pca_data[, !colnames(pca_data) %in% "type"]
```

```r
# Perform PCA with scaling and centering
pca_results <- prcomp(pca_data, center = TRUE, scale. = TRUE)

# Summary of PCA results
summary(pca_results)

# Explained variance for each component
cat("Explained variance:\n")
print(pca_results$sdev^2 / sum(pca_results$sdev^2) * 100)

# Plot of explained variance
fviz_eig(pca_results, addlabels = TRUE, ylim = c(0, 100))

# Project data onto the first two principal components
pca_projected <- as.data.frame(pca_results$x[, 1:2])
pca_projected$Type <- combined_wine$type

# Biplot visualization
fviz_pca_biplot(pca_results, geom = "point", habillage = combined_wine$type,
                addEllipses = TRUE, ellipse.level = 0.95)

# Extract the first five principal components
scores <- as.data.frame(pca_results$x[, 1:5])

# Elbow method to determine optimal number of clusters
set.seed(123)
wss <- sapply(1:10, function(k) {
  kmeans(scores, centers = k, nstart = 10, iter.max = 50)$tot.withinss
})

# Plot the Elbow method graph
plot(1:10, wss, type = "b", pch = 19, frame = FALSE,
     xlab = "Number of Clusters",
     ylab = "Total Within-Cluster Sum of Squares",
     main = "Elbow Method for the Optimal Number of Clusters")

# K-MEANS PCA

# Optimal number of clusters
optimal_clusters <- 4

# Perform KMeans clustering
set.seed(123)
kmeans_result <- kmeans(scores, centers = optimal_clusters, nstart = 10)

# Add clustering results to dataframe
scores_df <- data.frame(
  scores,
  Cluster = factor(kmeans_result$cluster),
  Type = combined_wine$type
)

# Convert 'Type' to factor
scores_df$Type <- as.factor(scores_df$Type)

# Plot using 'Type'
ggplot(scores_df, aes(x = PC1, y = PC2, color = Cluster, shape = Type)) +
  geom_point(alpha = 1, size = 3.5) +
  scale_color_manual(values = c("#dd1e2b", "#ff914d", "#5e17eb", "purple")) +
  scale_shape_manual(values = c(1, 3, 4)) +
  theme_minimal() +
  labs(title = "PCA with K-means Clustering",
       x = "Principal Component 1",
       y = "Principal Component 2",
       color = "Cluster",
       shape = "Type of Wine")

# Calculate silhouette score
silhouette_score <- silhouette(kmeans_result$cluster, dist(scores))

# Visualize the silhouette score
fviz_silhouette(silhouette_score)

# Calculate and print average silhouette width
average_silhouette <- mean(silhouette_score[, 3])
```

```r
cat("Average Silhouette Width: ", average_silhouette, "\n")

# Repeat KMeans with a different number of clusters (e.g., 2)
optimal_clusters <- 2
set.seed(123)
kmeans_result <- kmeans(scores, centers = optimal_clusters, nstart = 10)
scores_df$Cluster <- factor(kmeans_result$cluster)

# Updated plot
ggplot(scores_df, aes(x = PC1, y = PC2, color = Cluster, shape = Type)) +
  geom_point(alpha = 1, size = 3.5) +
  scale_color_manual(values = c("#dd1e2b", "#ff914d")) +
  scale_shape_manual(values = c(1, 3, 4)) +
  theme_minimal() +
  labs(title = "PCA with K-means Clustering",
       x = "Principal Component 1",
       y = "Principal Component 2",
       color = "Cluster",
       shape = "Type of Wine")

# Calculate silhouette score for the new clustering
silhouette_score <- silhouette(kmeans_result$cluster, dist(scores))
fviz_silhouette(silhouette_score)
average_silhouette <- mean(silhouette_score[, 3])
cat("Average Silhouette Width: ", average_silhouette, "\n")

# WARD2 Hierarchical Clustering

# Perform hierarchical clustering with "ward.D2" method
hc_ward <- hclust(dist(scores), method = "ward.D2")

# Plot the dendrogram
plot(hc_ward, main = "Hierarchical Clustering Dendrogram (ward.D2)", sub = "", xlab = "")

# Cut the dendrogram to obtain 4 clusters
clusters_hc_ward <- cutree(hc_ward, k = 4)

# Create dataframe with new clusters
pca_df_ward <- data.frame(
  scores,
  HC_Cluster = as.factor(clusters_hc_ward),
  Type = combined_wine$type
)

# Convert 'Type' to factor
pca_df_ward$Type <- as.factor(pca_df_ward$Type)

# Visualize new clusters
ggplot(pca_df_ward, aes(x = PC1, y = PC2, color = HC_Cluster, shape = Type)) +
  geom_point(alpha = 1, size = 3.5) +
  scale_color_manual(values = c("#dd1e2b", "#ff914d", "#5e17eb", "purple")) +
  scale_shape_manual(values = c(1, 3, 4)) +
  theme_minimal() +
  labs(
    title = "PCA with Hierarchical Clustering (Ward.D2)",
    x = "Principal Component 1",
    y = "Principal Component 2",
    color = "Cluster",
    shape = "Type of Wine"
  )

# Calculate silhouette score for hierarchical clustering
sil_scores_hc_ward <- silhouette(clusters_hc_ward, dist(scores))

# Visualize silhouette score
fviz_silhouette(sil_scores_hc_ward)

# Calculate and print average silhouette width
average_silhouette_hc_ward <- mean(sil_scores_hc_ward[, 3])
cat("Average Silhouette Width (Ward.D2): ", average_silhouette_hc_ward, "\n")

# K-MEANS UMAP

# Prepare data excluding 'type'
combined_wine_features <- combined_wine[, sapply(combined_wine, is.numeric)]
combined_wine_features <- combined_wine_features[, !colnames(combined_wine_features) %in% "type"]
```

```r
# Configure UMAP
umap_config <- umap.defaults
umap_config$n_neighbors <- 15
umap_config$min_dist <- 0.01
umap_config$n_components <- 2

# Apply UMAP to the data
umap_results <- umap(combined_wine_features, config = umap_config)

# Convert UMAP results to dataframe
umap_df <- as.data.frame(umap_results$layout)
colnames(umap_df) <- c("UMAP1", "UMAP2")

# Add 'Type' variable
umap_df$Type <- combined_wine$type

# Convert 'Type' to factor
umap_df$Type <- as.factor(umap_df$Type)

# Elbow method for optimal number of clusters on UMAP data
set.seed(123)
wss <- sapply(1:10, function(k) {
  kmeans(umap_df[, c("UMAP1", "UMAP2")], centers = k, nstart = 10)$tot.withinss
})

# Plot the Elbow method graph
plot(1:10, wss, type = "b", pch = 19, frame = FALSE,
     xlab = "Number of Clusters (K)",
     ylab = "Total Within-Cluster Sum of Squares",
     main = "Elbow Method for Choosing the Optimal Number of Clusters")

# Apply KMeans to UMAP results
optimal_clusters <- 3
kmeans_result <- kmeans(umap_df[, c("UMAP1", "UMAP2")], centers = optimal_clusters, nstart = 10)

# Add clustering results to dataframe
umap_df$Cluster <- as.factor(kmeans_result$cluster)

# Visualize UMAP clusters
unique_types <- length(levels(umap_df$Type))
shape_values <- c(1, 3, 4, 5, 6, 7, 8)[1:unique_types]

ggplot(umap_df, aes(x = UMAP1, y = UMAP2, color = Cluster, shape = Type)) +
  geom_point(alpha = 1, size = 3.5) +
  scale_color_manual(values = c("#dd1e2b", "#ff914d", "#5e17eb")) +
  scale_shape_manual(values = shape_values) +
  theme_minimal() +
  labs(
    title = "UMAP with KMeans Clustering",
    x = "UMAP Dimension 1",
    y = "UMAP Dimension 2",
    color = "Cluster",
    shape = "Type of Wine"
  )

# Calculate silhouette score for KMeans on UMAP results
sil_scores_kmeans <- silhouette(kmeans_result$cluster, dist(umap_df[, c("UMAP1", "UMAP2")]))
fviz_silhouette(sil_scores_kmeans)

# Calculate and print average silhouette width
average_silhouette_kmeans <- mean(sil_scores_kmeans[, 3])
cat("Average Silhouette Width: ", average_silhouette_kmeans, "\n")

# UMAP Hierarchical Clustering

# Hierarchical clustering on UMAP results
hc <- hclust(dist(umap_df[, c("UMAP1", "UMAP2")]), method = "complete")

# Plot the dendrogram
plot(hc, main = "Hierarchical Clustering Dendrogram (UMAP)", sub = "", xlab = "")

# Cut the dendrogram to obtain 3 clusters
clusters_hc <- cutree(hc, k = 3)

# Add clustering results to dataframe
```

```r
umap_df$HC_Cluster <- as.factor(clusters_hc)

# Visualize clusters using UMAP components
ggplot(umap_df, aes(x = UMAP1, y = UMAP2, color = HC_Cluster, shape = Type)) +
  geom_point(alpha = 1, size = 3.5) +
  scale_color_manual(values = c("#dd1e2b", "#ff914d", "#5e17eb")) +
  scale_shape_manual(values = c(1, 3)) +
  theme_minimal() +
  labs(
    title = "UMAP with Hierarchical Clustering",
    x = "UMAP Dimension 1",
    y = "UMAP Dimension 2",
    color = "Cluster",
    shape = "Type of Wine"
  )

# Calculate silhouette score for hierarchical clustering
sil_scores_hc <- silhouette(clusters_hc, dist(umap_df[, c("UMAP1", "UMAP2")]))
fviz_silhouette(sil_scores_hc)
average_silhouette_hc <- mean(sil_scores_hc[, 3])
cat("Average Silhouette Width (Hierarchical Clustering): ", average_silhouette_hc, "\n")

# Calculate average values for each variable within hierarchical clusters
combined_wine$HC_Cluster <- clusters_hc
relevant_columns <- colnames(combined_wine_features)
average_scores_hc <- aggregate(. ~ HC_Cluster, data = combined_wine[, c(relevant_columns, "HC_Cluster")], mean)
print(average_scores_hc)

# T-SNE KMEANS

# Remove duplicate rows from the dataset
combined_wine_unique <- unique(combined_wine_features)

# Find indices of remaining rows in the original combined dataset
unique_indices <- which(!duplicated(combined_wine_features))

# Get 'type' variable corresponding to non-duplicate rows
type_after_unique <- combined_wine$type[unique_indices]

# Perform t-SNE on data without 'type' and duplicates
set.seed(123)
tsne_results <- Rtsne(combined_wine_unique, dims = 2, perplexity = 30, verbose = TRUE, max_iter = 3000)

# Convert t-SNE results to dataframe
tsne_df <- as.data.frame(tsne_results$Y)
colnames(tsne_df) <- c("TSNE1", "TSNE2")

# Add 'type' variable to t-SNE dataframe
tsne_df$Type <- as.factor(type_after_unique)

# Perform KMeans clustering on t-SNE results
set.seed(123)
optimal_clusters <- 3
kmeans_tsne_result <- kmeans(tsne_df[, 1:2], centers = optimal_clusters, nstart = 20)
tsne_df$Cluster <- factor(kmeans_tsne_result$cluster)

# Visualize clusters using t-SNE components
ggplot(tsne_df, aes(x = TSNE1, y = TSNE2, color = Cluster, shape = Type)) +
  geom_point(alpha = 1, size = 3.5) +
  scale_color_manual(values = c("#dd1e2b", "#ff914d", "#5e17eb")) +
  scale_shape_manual(values = c(1, 3, 4)) +
  theme_minimal() +
  labs(
    title = "Clustering t-SNE with KMeans",
    x = "t-SNE Dimension 1",
    y = "t-SNE Dimension 2",
    color = "Cluster",
    shape = "Type of Wine"
  )

# Calculate silhouette score for t-SNE + KMeans
tsne_silhouette <- silhouette(kmeans_tsne_result$cluster, dist(tsne_df[, 1:2]))
fviz_silhouette(tsne_silhouette)
average_silhouette_tsne <- mean(tsne_silhouette[, 3])
cat("Average Silhouette Width (t-SNE + KMeans): ", average_silhouette_tsne, "\n")
```

```r
# Add KMeans clustering result to data without duplicates
combined_wine_unique$Cluster <- kmeans_tsne_result$cluster

# Calculate means for each cluster
cluster_means <- aggregate(. ~ Cluster, data = combined_wine_unique, mean)
print(cluster_means)

# T-SNE HIERARCHICAL

# Perform hierarchical clustering using "centroid" method
dist_tsne <- dist(tsne_df[, 1:2])
hc_tsne <- hclust(dist_tsne, method = "centroid")

# Plot the dendrogram
plot(hc_tsne, main = "Hierarchical Clustering Dendrogram (t-SNE)", sub = "", xlab = "")

# Cut the dendrogram to obtain 4 clusters
clusters_hc_tsne <- cutree(hc_tsne, k = 4)
tsne_df$HC_Cluster <- factor(clusters_hc_tsne)

# Visualize clusters using t-SNE components
ggplot(tsne_df, aes(x = TSNE1, y = TSNE2, color = HC_Cluster, shape = Type)) +
  geom_point(alpha = 1, size = 3.5) +
  scale_color_manual(values = c("#dd1e2b", "#ff914d", "#5e17eb", "purple")) +
  scale_shape_manual(values = c(1, 3, 4, 5)) +
  theme_minimal() +
  labs(
    title = "t-SNE with Hierarchical Clustering",
    x = "t-SNE Dimension 1",
    y = "t-SNE Dimension 2",
    color = "Cluster",
    shape = "Type of Wine"
  )

# Calculate silhouette score for hierarchical clustering
sil_scores_hc_tsne <- silhouette(clusters_hc_tsne, dist_tsne)
fviz_silhouette(sil_scores_hc_tsne)
average_silhouette_hc_tsne <- mean(sil_scores_hc_tsne[, 3])
cat("Average silhouette width (Hierarchical): ", average_silhouette_hc_tsne, "\n")
```