Final Project (Group 2)

Comparative Analysis of Machine Learning Techniques for Red Wine Quality Classification

Hongyi Zhang[1], Mengqi Liu[1], and Aoyu Liu[1]

[1] *McKelvey School of Engineering, Washington University in St. Louis, USA*
hongyi.zhang@wustl.edu, liumengqi@wustl.edu, aoyu@wustl.edu

Instructor: Professor Jinsong Zhang

1 Introduction

The objective of this project is to use machine learning algorithms to classify the quality of red wine, focusing on exploring the performance of different classification algorithms in this task. Machine learning is one of the core methods of modern data analysis, capable of predicting or classifying new data by learning patterns in existing data. By training and evaluating the model, we can determine the best way to improve prediction accuracy. This project selects the Wine Quality Dataset from the UCI machine learning database. The dataset contained 1,599 records of red wine, each of which included 11 chemical attributes (fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, and alcohol) and a target variable (quality).

The goal of this dataset is to predict the quality score of a red wine (ranging from 3 to 8). To simplify the task, we transform the original multi-classification problem into a binary classification problem, labeling samples with a quality score greater than or equal to 6 as 'high quality' (label 1) and samples with a quality score less than 6 as 'low quality' (label 0).

The objectives of the project include: (1) A variety of machine learning models, including decision tree, random forest, support vector machine, and artificial neural networks, were used to make classification predictions for red wine quality. (2) Compare the performance of different models in classification tasks and evaluate their accuracy, precision, recall, and F1-score. And analyze the advantages and disadvantages of these models. (3) Explore how hyperparameters influence model performance and investigate methods to improve model accuracy on the test set.

We used the following four classification algorithms in the project: (1) Decision Tree: By constructing a decision tree to classify data points. (2) Random Forest: Use multiple decision trees for ensemble learning to improve classification performance and reduce overfitting risks through voting mechanisms. (3) Support Vector Machine: Used for classification tasks that find the optimal hyperplane to maximize the margin between data points of different classes. (4) Artificial Neural Networks: Fully connected neural networks are used to model complex input-output relationships through nonlinear activation functions.

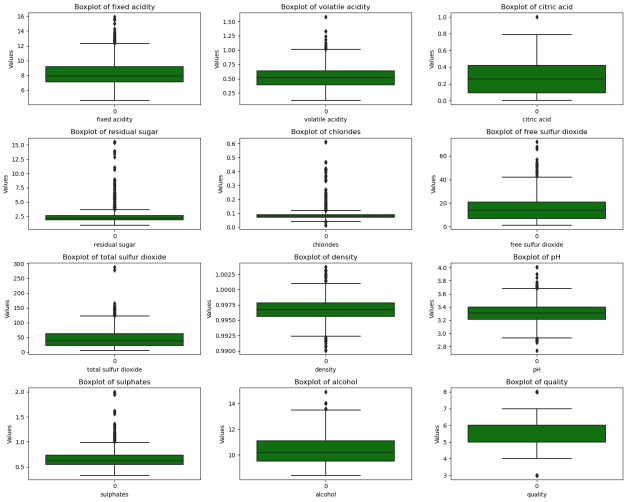
2 Exploratory Analysis of the Dataset

2.1 Missing Value and Outliers Detection

Before conducting data analysis and visualization, it is necessary to check for missing data to avoid issues during the analysis process. Upon inspection, it was found that there are no missing values in the dataset.

After performing the missing value check, the next step is to detect outliers in the data and visualize them using boxplots. Below are boxplots visualization of 12 numerical categories.

Figure 1: Boxplots of Red Wine Dataset



From the above figure, it can be observed that some features exhibit significant skewed distributions and outliers, such as residual sugar, chlorides, and sulphates. To reduce the impact of poor data distribution on the model, the StandardScaler method from the sklearn package will be used to standardize the training and test sets separately after splitting the data. This helps improve the data distribution. Standardizing after splitting the dataset is done to prevent data leakage and to enhance the robustness of the model.

2.2 Basic Statistical Analysis

The basic statistical analysis is a statistical inspection of the 12 numerical variables. The table below provides the statistical information for them, which provides some outlines of these data.

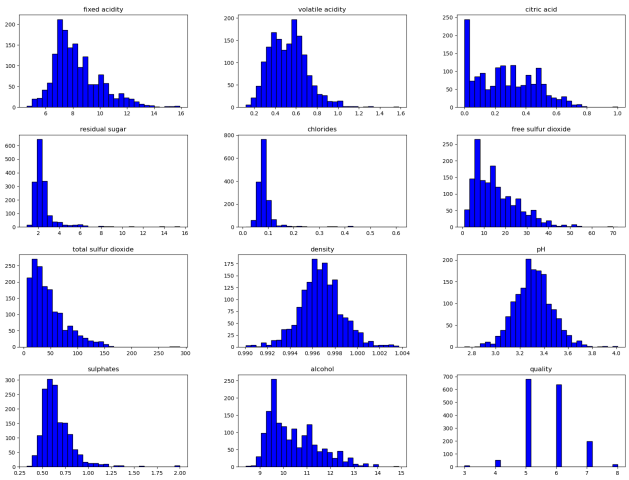
Figure 2: Basic Statistical Information about Features of Red Wine

Statistic	Fixed Acidity	Volatile Acidity	Citric Acid	Residual Sugar	Chlorides	Free Sulfur Dioxide	Total Sulfur Dioxide	Density	pH	Sulphates	Alcohol	Quality
Min	4.6	0.12	0.0	0.9	0.012	1.0	6.0	0.99	2.74	0.33	8.4	3.0
Max	15.9	1.58	1.0	15.5	0.611	72.0	289.0	1.004	4.01	2.0	14.9	8.0
Mean	8.32	0.53	0.27	2.54	0.09	15.87	46.47	0.997	3.31	0.66	10.42	5.64
Std	1.74	0.18	0.19	1.41	0.047	10.46	32.9	0.002	0.15	0.17	1.07	0.81

2.3 Data Distribution

Afterward, we can use histograms to observe the distribution of these data. The distribution plots are shown below.

Figure 3: Basic Statistical Information about features

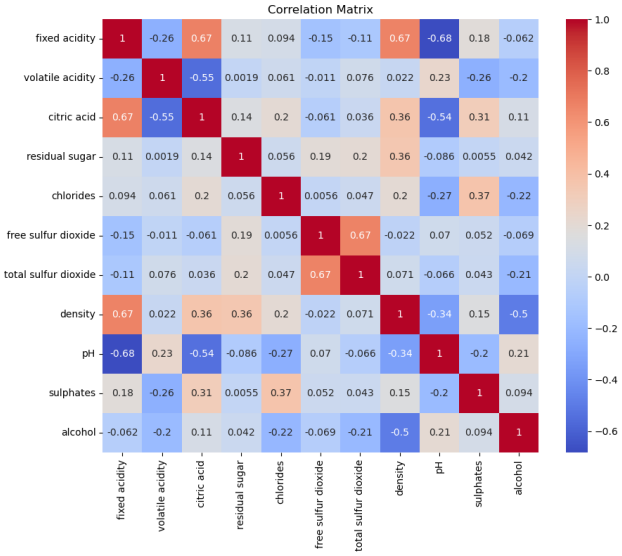


Combining the information from the above images and the above tables, we can visualize the overall status of the data more clearly.

2.4 Feature Correlation Analysis

To more intuitively observe the relationships between the feature values, we plotted the correlation matrix of the features. From the heatmap, it can be seen that the overall correlation between the features is relatively low, and therefore, dimensionality reduction is not necessary.

Figure 4: Correlation Matrix of Features



3 Methods

3.1 Decision Tree

Decision tree is a commonly used supervised machine learning method, widely used in classification problems. Its core is to divide the data step by step by a series of bifurcation splits or multifurcation splits based on features, always forming a decision path from the root node to the leaf nodes. In the classification task, the decision tree maximizes the purity of the leaf nodes by choosing the features and thresholds that best differentiate between the targets at each node. It calculates the impurity by computing the Entropy or Gini Index. Finally, calculate the Information Gain for different classifications and select the maximum case to determine the feature type and threshold for splitting.

3.1.1 Baseline Model of Decision Tree

In this study, we first trained a baseline decision tree model as a reference for benchmark performance. The baseline model used the default parameters of decision tree classifier. The criterion was set to "entropy" and the random state was set to 0.

During the construction of the baseline model, the training set was used for model fitting, and predictions were made on the test set to obtain the classification metrics of the baseline model, which contain overall accuracy, precision, recall, and F1-score.

3.1.2 Hyperparameter Tuning and Cross-validation of Decision Tree

Key hyperparameters included: (1) criterion: The function to measure the quality of a split (e.g., 'gini', 'entropy'). (2) max depth: The maximum depth of the tree, controlling its complexity and preventing overfitting. (3) min samples split: The minimum number of samples required to split an internal node, influencing the granularity of splits. (4) min samples leaf: The minimum number of samples required to form a leaf node, affecting the minimum size of terminal nodes.

To further enhance the predictive performance and generalization ability of the decision tree, we performed hyperparameter tuning using Grid Search and Cross-validation. By defining a parameter grid and using GridSearchCV to perform an exhaustive search, we evaluated the performance of each parameter combination on the training set using 5-fold cross-validation, allowing us to select the optimal parameter combination in terms of overall performance.

3.2 Random Forest

Random Forest is an ensemble learning method that improves overall predictive performance and robustness by combining the predictions of multiple decision tree models. During the training process, Random Forest employs bootstrap sampling and random feature subset selection to construct a set of independent and diverse decision trees. The final classification result is determined through voting on the predictions of individual trees. Compared to a single decision tree, Random Forest typically offers higher generalization ability and stability, and demonstrates strong resistance to overfitting.

3.2.1 Baseline Model of Random Forest

In this study, we trained a baseline Random Forest model using the default parameters of Random Forest Classifier. After training the model on the training set, we evaluated its performance on the test set, including metrics such as overall accuracy, precision, recall, and F1-score. This step provides a reference for comparison in subsequent hyperparameter tuning.

3.2.2 Hyperparameter Tuning and Cross-validation of Random Forest

Key hyperparameters included: (1) n estimators: The number of trees in the forest, controlling the ensemble size. (2) max depth: The maximum depth of the decision trees, determining the complexity of the individual trees. (3) min samples split: The minimum number of samples required to split an internal node, influencing the granularity of the splits.

To further improve the predictive performance of the Random Forest model, we conducted a Grid Search combined with 5-fold cross-validation to tune its key hyperparameters. Specifically, we systematically searched for optimal values of parameters such as the number of trees in the forest, the maximum depth of the decision trees, and the minimum number of samples required to split an internal node.

After identifying the optimal parameter combination, we used the best hyperparameters to construct the optimized model. This model outperformed the baseline model on the test set, demonstrating the significant impact of appropriate hyperparameter tuning on performance improvement.

3.3 Support Vector Machine

In this project, the Support Vector Machine (SVM) was employed as a classification model to predict red wine quality. The primary objective of SVM is to identify an optimal hyperplane to maximize the margin that effectively separates data points from different classes. To enhance the model's generalization ability, both a baseline SVM model and a hyperparameter-tuned model were implemented.

3.3.1 Baseline Model of SVM

3.3.2 Hyperparameter Tuning and Cross-validation of SVM

Key hyperparameters included: (1) c: The regularization parameter, controlling the flexibility of the classification boundary. (2) kernel: The type of kernel function (e.g., 'rbf', 'poly', 'linear'). (3) gamma: Kernel coefficient for non-linear kernels. (4) degree: Degree of the polynomial kernel.

Grid search with cross-validation (GridSearchCV) was employed (cv=5) to identify the optimal parameter combination on the training set. The best parameters were extracted and used as the model's hyperparameters. The model was then retested with these parameters and evaluated on the test set.

3.4 Artificial Neural Networks

In this project, the Multi-Layer Perceptron (MLP) was implemented to perform classification tasks. A baseline MLP model was implemented first to see its basic ability, and based on it a model after hyperparameter tuning was implemented to reach a better accuracy of classification tasks.

3.4.1 Baseline Model of ANN

An initial MLP Classifier model was created with default parameters, a maximum iteration of 2000, and a fixed random state for reproducibility. The model was trained on the training dataset and the evaluations were generated based on the test set. To evaluate the performance, accuracy, precision, recall, F1-score and confusion matrix were applied to have a comprehensive result.

3.4.2 Hyperparameter Tuning and Cross-validation of ANN

A detailed GridSearchCV approach was employed to optimize the hyperparameters of the MLPClassifier. The parameters tuned included: (1) hidden layer sizes: The structure of the hidden layers in the neural network. Adjusting this parameter can control the complexity and expressiveness of the model. (2) activation: Define the activation function used in the hidden layer. In the project the Rectified Linear Unit function was chosen. (3) solver: Define optimization algorithm for weight update. In the project Adam algorithm was chosen. (4) alpha: The L2 regularization parameter used to prevent overfitting. (5) learning rate: The learning rate adjustment strategy defines how the learning rate changes dynamically. In the project adaptive strategy was chosen to automatically adjust the learning rate based on training progress. (6) learning rate init: The initial learning rate controls the step size of the weight update.

Based on these hyperparameters GridSearchCV was implemented to find the best parameter combination. Similar evaluations as above.

4 Results and Analysis

4.1 Decision Tree

For the baseline decision tree model, the precision is 0.72, the recall is 0.74, the F1-score is 0.73 and the accuracy is 0.7469 for low-quality red wine.

After performing hyperparameter tuning using Grid Search with 5-fold cross-validation, the best hyperparameters for the model were identified as criterion equal to entropy, max depth equal to 6, min samples leaf equal to 12, and min samples split equals 2. Based on this tuned model, for low-quality red wine, the precision is 0.74, the recall is 0.72, the F1-score is 0.73, and the accuracy is 0.7531

Compared with the two models, the accuracy is increased from 0.7469 to 0.7531, indicating that adjusting the hyperparameters and cross-validation can improve the performance of the model

4.2 Random Forest

For the baseline random forest model, we can calculate that the precision is 0.80, the recall is 0.82, the F1-score is 0.81, and the accuracy is 0.8219 of low-quality red wine.

After performing hyperparameter tuning using Grid Search with 5-fold cross-validation, the best hyperparameters for the model were identified as max depth equal to none, min samples split equal to 2, and number of estimators equals to 200. Based on this model, for low-quality red wine, the precision is 0.80, the recall is 0.82, the F1-score is 0.81, and the accuracy is 0.8250.

Compared with the two models, the accuracy is increased from 0.8219 to 0.8250, indicating that adjusting the hyperparameters and cross-validation can improve the performance of the model

4.3 Support Vector Machine

For the baseline SVM model, we can calculate that the precision is 0.75, the recall is 0.74, the F1-score is 0.74, and the accuracy is 0.7625 for low-quality red wine class.

The best parameters identified were: c set to 5, kernel set to 'rbf', gamma set to 0.1, and degree set to 2. The tuned model achieved a classification accuracy of 0.7781. The F1-score for Class 0 improved to 0.76, and for Class 1, it increased to 0.80. The macro-average and weighted-average accuracy rose to 0.78.

The tuned model's accuracy improved by 1.56% compared to the baseline model. Metrics such as F1-score and recall also demonstrated improvements. The confusion matrix indicates that the tuned model performed better in identifying high-quality wine, with fewer classification errors.

4.4 Artificial Neural Networks

For the baseline model, the precision is 0.77, the recall is 0.71, the F1-score is 0.74, and the accuracy is 0.7656 for low-quality red wine.

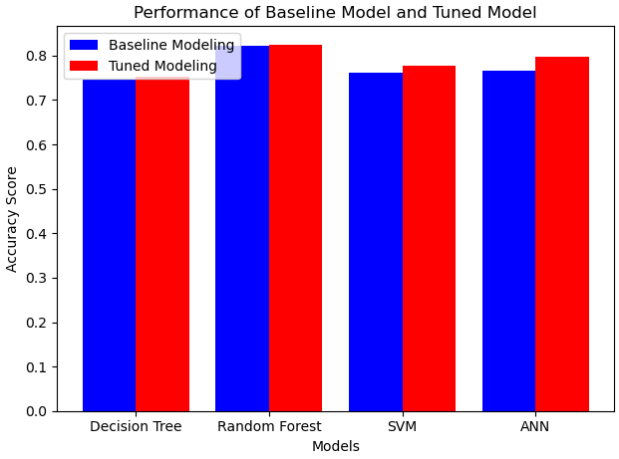
The optimal parameter combination was identified as alpha set to 0.001, hidden layer sizes configured as (100, 100), and the initial learning rate set to 0.01. The accuracy of the MLP model reached 79.69% after tuning.

For the two classes, all three metrics were improved to 0.80, both in macro-average and in weighted-average. Compared with the baseline model, for Class 0, all three metrics improved. For Class 1, although the recall dropped to 0.03, the rest of the metrics were better as well.

4.5 Summary and Analysis

To simplify the comparison, we directly selected the simplest and most intuitive metric, accuracy, as the evaluation standard. The figure below shows the accuracy of the baseline and tuned models for each algorithm. This figure demonstrates that using GridSearchCV for hyperparameter tuning and cross-validation can effectively improve model performance. Among them, the tuned Random Forest model achieved the highest accuracy of 0.8250, highlighting that ensemble learning can enhance model performance.

Figure 5: Performance of Baseline Models and Tuned Models



5 Conclusion

Through the previous results, it is shown that among the four methods, Random Forest had the best performance, with an accuracy of 0.8250. Compared with the baseline models, all the models had improvements after the hyperparameter tuning and cross-validation, which shows the importance of choosing proper hyperparameters for different models. Furthermore, the results demonstrates that ensemble learning can effectively enhance model performance from comparation between decision tree and random forest. And it also indicates that more complex models may have better performance, but overfitting can also occur, leading to a decrease in accuracy. Even though in this project the target values were divided only into two classes in order to have a higher accuracy, it could be further analyzed in the future to see the reason why the original target values had a bad performance and consider a wider range of hyperparameters to find better parameter combinations for higher accuracy.

Final Project Work Distribution:

Hongyi Zhang: Exploratory Analysis of the Dataset, Decision Tree and Random Forest, and LaTeX Typesetting. **Mengqi Liu**: Introduction, Support Vector Machine. **Aoyu Liu**: Conclusion, Artificial Neural Network.

6 Appendix

The following section is the Python code for the project.

```
# Import necessary libraries

# basic libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats

# sklearn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

# models
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
```

6.1 Data Loading and Preprocessing

```
# Load the dataset
data_initial = pd.read_csv('winequality-red-4.csv')
# data_initial.head(5)

# Split the header and the data
header = data_initial.columns[0].replace('"', '').split(';')

# Split the data
data = data_initial.iloc[:,0].str.split(';', expand=True)

# Transfrom the string data to float
data = data.astype(float)

# Assemble the data
data.columns = header
# data.head(5)
```

6.2 Exploratory Analysis of the Dataset

6.2.1 Basic Statistical Analysis

```
# Basic statistics
data.describe()

# Creating the DataFrame
df = pd.DataFrame(data.describe().loc[['mean', 'std', 'min', 'max'], :].applymap(lambda x:
    f"{x:.3f}"))
```

```
# Make the index the first column
df.insert(0, 'Statistics', ['mean', 'std', 'min', 'max'])

# Plotting the table as an image
fig, ax = plt.subplots(figsize=(14, 6))
ax.axis('off')

# Create a table with adjusted column widths
table = ax.table(cellText=df.values, colLabels=df.columns, cellLoc='center', loc='center')

# Styling the table
table.auto_set_font_size(False)
table.set_fontsize(10)

# Adjust column widths manually for better spacing
col_widths = [1] + [0.3] * (len(df.columns) - 1)
for i, width in enumerate(col_widths):
    table.auto_set_column_width(i)
    for key, cell in table.get_celld().items():
        if key[1] == i:
            cell.set_width(width)

# Adjust the row heights
for key, cell in table.get_celld().items():
    cell.set_height(0.05)

# Save the adjusted table as an image
plt.savefig("./images/statistics_table.png", bbox_inches='tight', dpi=1000)
plt.show()
```

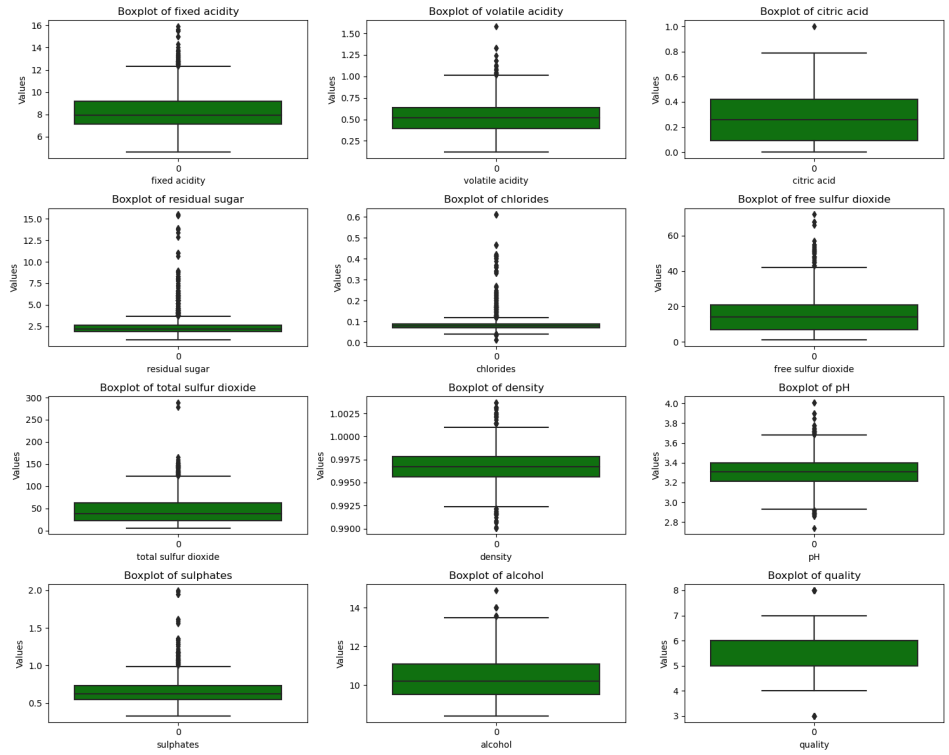
Statistic	Fixed Acidity	Volatile Acidity	Citric Acid	Residual Sugar	Chlorides	Free Sulfur Dioxide	Total Sulfur Dioxide	Density	pH	Sulphates	Alcohol	Quality
Min	4.6	0.12	0.0	0.9	0.012	1.0	6.0	0.99	2.74	0.33	8.4	3.0
Max	15.9	1.58	1.0	15.5	0.611	72.0	289.0	1.004	4.01	2.0	14.9	8.0
Mean	8.32	0.53	0.27	2.54	0.09	15.87	46.47	0.997	3.31	0.66	10.42	5.64
Std	1.74	0.18	0.19	1.41	0.047	10.46	32.9	0.002	0.15	0.17	1.07	0.81

6.2.2 Missing Value and Outliers Detection

```
# Check for missing values
data.isnull().sum()
```

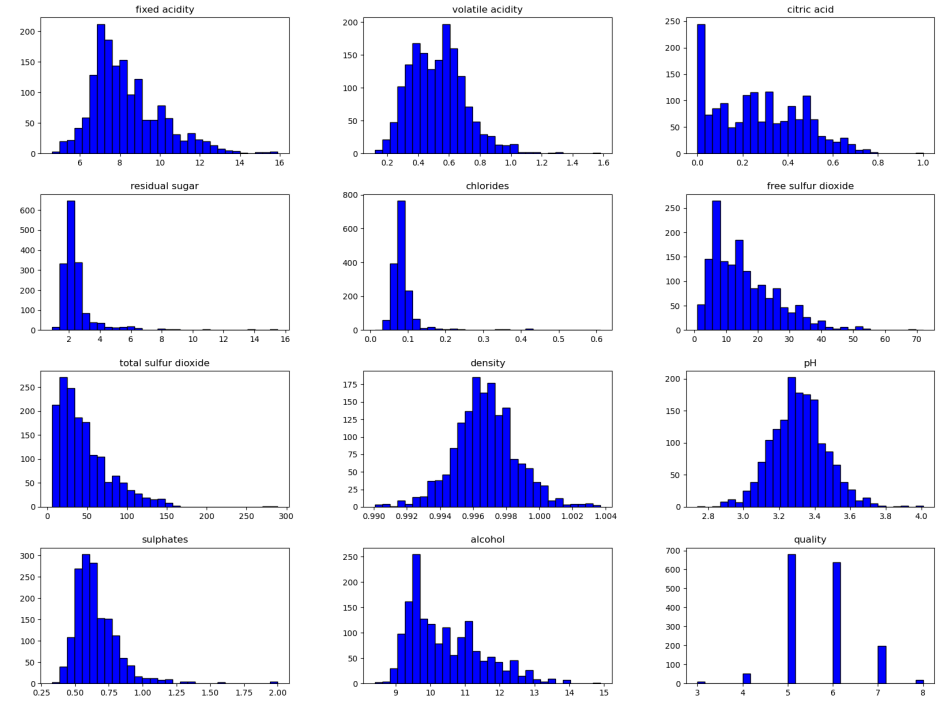
fixed acidity 0 volatile acidity 0 citric acid 0 residual sugar 0 chlorides 0 free sulfur dioxide 0 total sulfur dioxide 0 density 0 pH 0 sulphates 0 alcohol 0 quality 0 dtype: int64

```
# Boxplot of the data
fig, axes = plt.subplots(4, 3, figsize=(15, 12))
axes = axes.flatten()

for i, column in enumerate(data.columns):
    sns.boxplot(data[column], color='g', ax=axes[i])
    axes[i].set_title(f"Boxplot of {column}")
    axes[i].set_xlabel(column)
    axes[i].set_ylabel("Values")

plt.tight_layout()
plt.show()
```



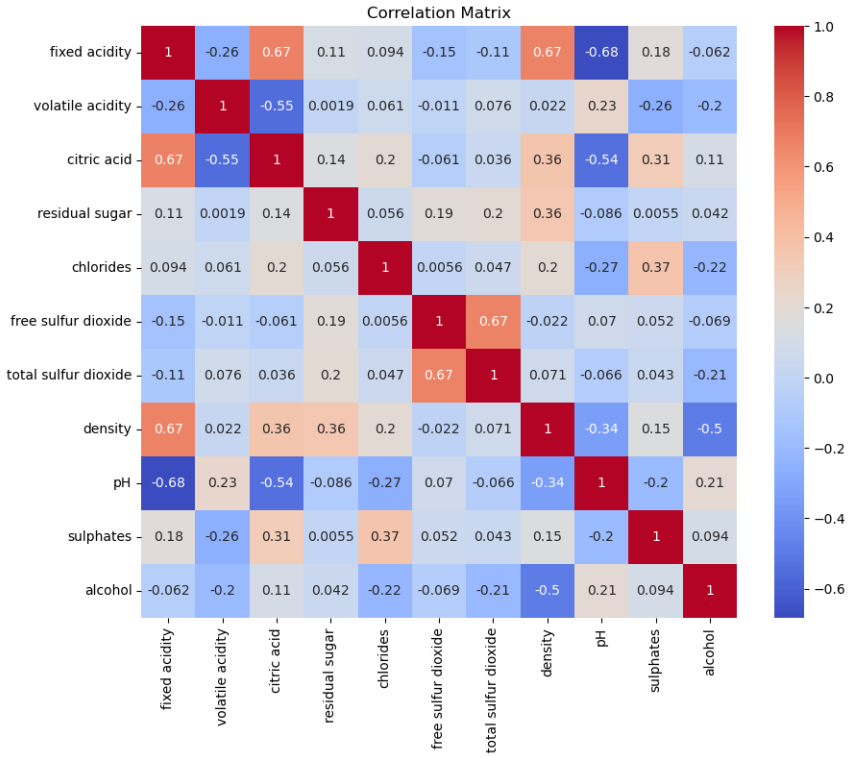
6.2.3 Data Distribution

```
# Plot all the distributions of these features use colorful histograms
data.hist(bins=30, figsize=(20, 15), color='b', edgecolor='black', linewidth=1.0, grid=False)
plt.show()
```



6.2.4 Feature Correlation Analysis

```
# Plot the correlation matrix
plt.figure(figsize=(10, 8))
sns.heatmap(data.iloc[:, :-1].corr(), annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()
```



6.3 Construction of Target Values, Data Spliting, and Standardization

```
# Divide the quality into two classes, low quality (0) and high quality (1), based on the
    threshold 6
data['target'] = data['quality'].apply(lambda x: 1 if x >= 6 else 0)
# data.head(5)

# Separate the features and the target
X = data.drop(['quality', 'target'], axis=1)
y = data['target']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)

# Data standardization
# We must standardize the data after splitting the data into training and testing sets to avoid
    data leakage.
standard_scaler = StandardScaler()
X_train = standard_scaler.fit_transform(X_train)
X_test = standard_scaler.transform(X_test)
```

6.4 Training Model

6.4.1 Decision Tree

```
# Decision Tree Classifier - Baseline model
dt_baseline = DecisionTreeClassifier(random_state=0, criterion='entropy')
dt_baseline.fit(X_train, y_train)

y_pred_baseline = dt_baseline.predict(X_test)
decision_tree_baseline_accuracy = accuracy_score(y_test, y_pred_baseline)

print("Baseline Decision Tree Accuracy:", accuracy_score(y_test, y_pred_baseline))
print(classification_report(y_test, y_pred_baseline))
print(confusion_matrix(y_test, y_pred_baseline))
```

```
# Decision Tree Classifier - Hyperparameter tuning and cross-validation
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 2, 4, 6, 8, 10],
    'min_samples_split': [2, 4, 8, 16],
    'min_samples_leaf': [1, 2, 4, 8, 10, 12],
}

grid_search = GridSearchCV(dt_baseline, param_grid, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)

print("Best Params from GridSearch:", grid_search.best_params_)
print("Best CV Score from GridSearch:", grid_search.best_score_)

dt_best = grid_search.best_estimator_
y_pred_best = dt_best.predict(X_test)
decision_tree_best_accuracy = accuracy_score(y_test, y_pred_best)

print("Best Decision Tree Accuracy:", accuracy_score(y_test, y_pred_best))
print(classification_report(y_test, y_pred_best))
print(confusion_matrix(y_test, y_pred_best))
```

6.4.2 Random Forest

```
# Random Forest Classifier - Baseline Model
rf_baseline = RandomForestClassifier(random_state=0)
rf_baseline.fit(X_train, y_train)

y_pred_baseline = rf_baseline.predict(X_test)
random_forest_baseline_accuracy = accuracy_score(y_test, y_pred_baseline)

print("Baseline Random Forest Accuracy:", accuracy_score(y_test, y_pred_baseline))
print(classification_report(y_test, y_pred_baseline))
print(confusion_matrix(y_test, y_pred_baseline))
```

```
# Random Forest Classifier - Hyperparameter tuning and cross-validation
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 4, 8]
```

```
}

grid_search = GridSearchCV(rf_baseline, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train, y_train)

print("Best Params from GridSearch:", grid_search.best_params_)
print("Best CV Score from GridSearch:", grid_search.best_score_)

rf_best = grid_search.best_estimator_
y_pred_best = rf_best.predict(X_test)
random_forest_best_accuracy = accuracy_score(y_test, y_pred_best)

print("Best Random Forest Accuracy:", accuracy_score(y_test, y_pred_best))
print(classification_report(y_test, y_pred_best))
print(confusion_matrix(y_test, y_pred_best))
```

6.4.3 Support Vector Machine

```
# Support Vector Machine Classifier - Baseline Model
svm_baseline = SVC(random_state=0)
svm_baseline.fit(X_train, y_train)

y_pred_baseline = svm_baseline.predict(X_test)
svm_baseline_accuracy = accuracy_score(y_test, y_pred_baseline)

print("Baseline SVM Accuracy:", accuracy_score(y_test, y_pred_baseline))
print(classification_report(y_test, y_pred_baseline))
print(confusion_matrix(y_test, y_pred_baseline))
```

```
# Support Vector Machine Classifier - Hyperparameter tuning, regularization, and cross-validation
param_grid = {
    'C': [0.1, 1, 5],
    'gamma': [1, 0.1, 0.01],
    'kernel': ['rbf', 'poly', 'linear'],
    'degree': [2, 3, 4]
}

grid_search_svc = GridSearchCV(svm_baseline, param_grid, cv=5, scoring='accuracy', n_jobs=-1,
    error_score=0)
grid_search_svc.fit(X_train, y_train)

print("Best Params from GridSearch for SVM:", grid_search_svc.best_params_)
print("Best CV Score from GridSearch for SVM:", grid_search_svc.best_score_)

svm_best = grid_search_svc.best_estimator_
y_pred_best = svm_best.predict(X_test)
svm_best_accuracy = accuracy_score(y_test, y_pred_best)

print("Best SVM Accuracy:", accuracy_score(y_test, y_pred_best))
print(classification_report(y_test, y_pred_best))
print(confusion_matrix(y_test, y_pred_best))
```

6.4.4 Artificial Neural Networks

```
# Multi-Layer Perceptron Classifier - Baseline Model
mlp_baseline = MLPClassifier(max_iter=2000, random_state=0)
mlp_baseline.fit(X_train, y_train)

y_pred_baseline = mlp_baseline.predict(X_test)
ann_baseline_accuracy = accuracy_score(y_test, y_pred_baseline)

print("Baseline MLP Accuracy:", accuracy_score(y_test, y_pred_baseline))
print(classification_report(y_test, y_pred_baseline))
print(confusion_matrix(y_test, y_pred_baseline))
```

```
# Multi-Layer Perceptron Classifier - Hyperparameter tuning, regularization, and cross-validation
param_grid = {
    'hidden_layer_sizes': [(50,), (100,), (150,),
                           (50, 50), (100, 50), (100, 100)],
    'activation': ['relu'],
    'solver': ['adam'],
    'alpha': [0.0001, 0.001, 0.01],
    'learning_rate': ['adaptive'],
    'learning_rate_init': [0.001, 0.01]
}

mlp = MLPClassifier(max_iter=1000, early_stopping=True, random_state=0)

grid_search = GridSearchCV(
    estimator=mlp,
    param_grid=param_grid,
    scoring='accuracy',
    cv=5,
    verbose=2,
    n_jobs=-1
)

grid_search.fit(X_train, y_train)

print("Best parameter:", grid_search.best_params_)
print("Best cross validation accuracy", grid_search.best_score_)

best_mlp = grid_search.best_estimator_
y_pred_best = best_mlp.predict(X_test)
ann_best_accuracy = accuracy_score(y_test, y_pred_best)

print("Best ann accuracy", accuracy_score(y_test, y_pred_best))
print(classification_report(y_test, y_pred_best))
print(confusion_matrix(y_test, y_pred_best))
```

6.5 Results and Analysis

```
# Construct a dictionary to store the accuracy of each model
results = {'Decision Tree': [decision_tree_baseline_accuracy, decision_tree_best_accuracy],
           'Random Forest': [random_forest_baseline_accuracy, random_forest_best_accuracy],
           'SVM': [svm_baseline_accuracy, svm_best_accuracy],
           'ANN': [ann_baseline_accuracy, ann_best_accuracy]}

# Transform the dictionary to a list
models = list(results.keys())
```

```
normal_model = [results[model][0] for model in models]
tuned_model = [results[model][1] for model in models]

# Plot the barchart
x = np.arange(len(models))
width = 0.4
fig, ax = plt.subplots()
bars1 = ax.bar(x - width/2, normal_model, width, label='Baseline Modeling', color='b')
bars2 = ax.bar(x + width/2, tuned_model, width, label='Tuned Modeling', color='r')

# Add some text for labels, title and custom x-axis tick labels, etc.
ax.set_xlabel('Models')
ax.set_ylabel('Accuracy Score')
ax.set_title('Performance of Baseline Model and Tuned Model')
ax.set_xticks(x)
ax.set_xticklabels(models)
ax.legend()

plt.tight_layout()
plt.show()
```