Predicting Wine Quality Using Comparative Feature Selection Techniques

Jonah Keller (jnk52), Aryan Khanna (ak2253), Christopher Ambrus (caa66),

Haijiao Tao (ht485), and Jeana Hoffmann (jmh472)

Cornell University

STSCI 4740: Data Mining and Machine Learning

Dr. Yang Ning

May 8th, 2023

1. Introduction

The wine industry has always been interested in understanding the factors that contribute to wine quality. Accurate classification of wine quality is valuable to the wine industry, as it helps producers improve their products, enables vendors to make informed decisions when selecting wines to sell, and assists purchasers in choosing wines that meet their taste preferences. In this report, we analyze a dataset containing the chemical properties of red and white wines to determine whether these properties can be used to predict wine quality.

The dataset includes the following predictors, which represent various chemical properties of the wines: (1) Fixed acidity: Represents the concentration of non-volatile acids in the wine, primarily tartaric acid. (2) Volatile acidity: Refers to the concentration of acetic acid, which can give the wine an unpleasant vinegar taste at high levels. (3) Citric acid: A weak organic acid that can add freshness and flavor to wines. (4) Residual sugar: The amount of sugar remaining after fermentation, affects the sweetness of the wine. (5) Chlorides: Represents the concentration of salt in the wine. (6) Free sulfur dioxide: The amount of unbound SO2, which acts as a preservative and antioxidant. (7) Total sulfur dioxide: The sum of free and bound SO2, used to control oxidation and microbial growth in wine. (8) Density: The mass per unit volume of the wine, influenced by alcohol, sugar, and other dissolved solids. (9) pH: Indicates the acidity or basicity of the wine, affecting flavor and stability. (10) Sulfates: Contribute to the concentration of SO2, preserving freshness and protecting wine from oxidation. (11) Alcohol: The percentage of alcohol by volume, affects the body, flavor, and perception of sweetness in the wine.

For simplicity, we transform this into a **binary classification** problem, where wines with a quality score of **6 or above** are considered **good quality**, and wines with a score **below 6** are considered **poor quality**. Our goal is to predict whether a wine is of good or poor quality as best as possible. To accomplish this, we first explore the all-wine dataset and then employ a pipeline to find optimal machine learning (ML) models to classify wine quality based on feature subsets. The insights gained from this analysis can provide valuable guidance for the wine industry in terms of production, marketing, and purchasing decisions.

2. Methods

2.1 Pre-Processing

We begin the pre-processing phase by loading the all-wine dataset. First, we split the dataset into a **training set** and a **validation** (hold-out) **set** using a **70:30 split** (Training: Validation), **stratified** based on **wine type and wine quality**, ensuring their equal representation. The validation set approach is valuable in machine learning analysis because it allows us to assess the performance of our machine learning models on *unseen* data, ensuring that our models can generalize well to new data without overfitting. Again, stratification helps maintain the distribution of wine types and quality levels in both sets, preventing sampling biases that may skew our model's results.

Next, we plot the **histogram** of the quantitative quality variable on the training set and use it to define a binary cutoff: quality ≥ 6 indicates good quality, while quality < 6 indicates poor quality. We then visualize the distribution of the quantitative predictors in the all-wine dataset of the training set using histograms (see Figure 1 in appendix). **Quantile-quantile (Q-Q) plots** on each quantitative predictor are employed to evaluate whether **the data is not normally distributed** for the training data (see Figure 2), which appears to be the case—the red line indicates perfect normality. Non-normal distribution can be limiting when applying ML analysis, as it may impact the performance of certain models which assume normality.

To address this issue, we implement a power transform, a **Box-Cox transformation**, which aids in normalizing dependent variables for model fitting. Calculating the **lambda λ** for the Box-Cox transformation on the training set, we apply the transformation with the calculated λ to both the training and validation sets. We then replot the **histograms** and **Q-Q plots**, while also adding a **correlation matrix** and **box plots** for each predictor (see Figures 3, 4, 5, and 6), observing that the transformation restored normality. On the transformed training data, we **perform principal components analysis (PCA)** and plot the results (see Figures 7 and 8), shading the transformed points once based on quality and separately based on wine type. Upon inspection of the wine type plot, it became evident that the majority of the variance in the all-wine dataset is related to *wine type* rather than the other predictors. This observation served as justification for **splitting the dataset** based on wine type and performing separate analyses (see Figure 0).

Consequently, we repeated the pre-processing steps separately for both the white-wine and red-wine datasets. For each wine type, we start by creating train-test splits that are stratified based on wine quality. Then, we use histograms and Q-Q plots (see Figures 9 and 10 for white wine and Figures 16 and 17 for red wine) to **assess the normality of the data**. To address any non-normality, we calculate Box-Cox transformations on the training set and apply them to both the training and validation sets. After the transformation, we re-visualize the data using histograms, Q-Q plots, correlation matrices, and box plots (see Figures 11, 12, 13, 14, 18, 19, 20, and 21) to confirm that normality has been achieved. Lastly, we employ **PCA** to visualize the variance in the datasets, specifically focusing on wine quality (see Figures 15 and 22).

2.2 Machine Learning Analysis

We employed several different feature selection algorithms. Namely, **Forward Stepwise Selection (FSS)**, **Backward Stepwise Selection (BSS)**, **Lasso**, and **Recursive Feature Elimination (RFE)**.

First, we performed **FSS** for **Logistic Regression (LR)**. This process starts with an empty model and sequentially adds predictor variables. In each stage, the variable that most improves the model fit is added (based on AIC). This method is computationally efficient for high n and low p datasets, as it avoids testing all possible feature combinations. However, it might miss the optimal model due to its greedy nature. Next, we executed **BSS** for LR. This approach begins with a model containing all predictor variables and removes them one by one. At each step, the proposed model that improves the model fit (based on AIC/deviance) is chosen. The final model with the lowest AIC is selected. This method, like FSS, works well for high n, low p datasets, but also may not consider the best model due to its greediness. Thirdly, we employed **Lasso**, a shrinkage method that selects the model based on the goodness-of-fit according to the maximum likelihood function and a penalty term (weighted by lambda). The penalty term discourages the use of redundant or extra predictors, while the goodness-of-fit increases as predictors are added. This method is advantageous for high n and low p datasets because it can perform feature selection and regularization simultaneously, thus reducing overfitting. However, Lasso might struggle with high multicollinearity among predictors.

Lastly, we utilize **RFE with Cross-Validation** (RFECV) to pinpoint the most suitable set of features for our models. RFE is a powerful technique that iteratively **removes the least important features** from the dataset while training the model and assessing its performance. This process aids in identifying a subset of features that contribute significantly to the prediction task. RFE offers several advantages over other feature selection methods. Firstly, RFE can effectively identify and eliminate irrelevant or redundant features, thereby reducing the risk of overfitting and improving model interpretability. Additionally, **RFE considers the interaction between features** during the elimination process, providing a more comprehensive understanding of feature importance within the context of the entire dataset. This holistic approach to feature selection allows RFE to outperform other algorithms that may focus only on individual feature importance, potentially overlooking essential interactions that could impact model performance.

In our analysis, we employ **repeated k-fold cross-validation (CV)** in conjunction with **RFE** to guarantee a robust and less overfit-prone feature selection process. Repeated CV involves executing a k-fold CV several times and averaging the results, which offers a **more dependable** estimate of model performance compared to a single k-fold CV. The primary advantage of repeated k-fold CV over a single k-fold CV is its ability to **mitigate the impact of random sampling variability** on model performance estimates. By repeating the CV process, we obtain a more stable and reliable assessment of how well our model generalizes to unseen data. This improved reliability is especially crucial in the context of selecting the best model from a training set, as it helps us identify the optimal model with higher confidence. Compared to using a validation set, repeated k-fold CV provides a more efficient use of the available data. In a validation set approach, a portion of the data is held out for model evaluation and not used during training, which could lead to suboptimal model performance, especially with smaller datasets. On the other hand, k-fold CV utilizes the entire dataset for both training and evaluation, mitigating the risk of underutilizing valuable data. Lastly, employing any form of cross-validation is superior to not using cross-validation at all, as it helps to **prevent overfitting** and provides a more accurate estimate of how well the model will perform on unseen data. This becomes particularly important when selecting the best set of features and determining the most suitable model for our prediction task.

The RFE algorithm accepts a machine learning evaluator with importance or coefficient parameters. To minimize bias, we utilized various machine learning evaluators in our analysis, each with unique pros and cons:

1. **Logistic Regression (LR)**: A simple, interpretable linear model for binary classification tasks. It may struggle with complex feature-target relationships but performs well on linearly separable data.
2. **Linear Discriminant Analysis (LDA)**: A method for dimensionality reduction and classification. It assumes Gaussian distributions and may underperform when this assumption doesn't hold.
3. **Linear Support Vector Machine (SVM)**: A powerful model for linearly separable data but may struggle with non-linear relationships. Careful hyperparameter tuning is required.
4. **Ridge Classifier**: A linear model using L2 regularization to prevent overfitting. It may not perform well with non-linear relationships.
5. **Random Forest (RF)**: An ensemble model based on decision trees, effective in handling non-linear relationships and feature interactions. It requires more computational resources and may be harder to interpret.
6. **Gradient Boosting Trees (GBT)**: An ensemble model boosting weak learners, capable of handling non-linear relationships but potentially more prone to overfitting and computationally intensive.
7. **Decision Trees**: A simple, interpretable model capturing nonlinear relationships but prone to overfitting and potentially poor generalization to unseen data.
8. **Perceptron**: A simple linear model for binary classification tasks, which may struggle with complex feature-target relationships.
9. **Extra Trees Classifier (ETC)**: An ensemble model based on decision trees, handling non-linear relationships but requiring more computational resources and potentially harder to interpret.

Considering the high n, low p dataset for binary classification without interpretability requirements, ensemble methods such as RF and GBT are likely to perform well, as they can handle non-linear relationships and feature interactions. However, the choice of the best model ultimately depends on the specific dataset characteristics and the performance metrics of interest.

Post-**RFECV algorithm**, each evaluator has an optimal set of features based on **area under the curve (AUC)**. For each evaluator, we train a final model with those features using **repeated k-Fold CV**. Using the trained model, we then classify the quality of the validation set. From these predictions, we compute several classification metrics, including ***kappa***, ***precision***, ***accuracy***, ***recall/sensitivity***, ***AUC***, and ***F-score***, to assess model performance. Each metric has its pros and cons in assessing model performance, as they emphasize different aspects of the predictions. *Kappa* measures the agreement between predicted and actual classifications, accounting for the possibility of random agreement. It is particularly useful when the class distribution is imbalanced. *Precision* evaluates the proportion of true positive predictions out of all positive predictions made. It is essential when false positives are undesirable, such as in medical diagnoses. *Accuracy* measures the proportion of correct predictions out of all predictions made. It provides a general sense of model performance but can be misleading when the class distribution is imbalanced. *Recall*, also known as *sensitivity*, measures the proportion of true positive predictions out of all actual positive cases. It is crucial when detecting true positive cases is more important, such as in fraud detection. *AUC* provides an aggregate measure of a model's performance across various classification thresholds. It is useful for comparing models irrespective of the specific decision threshold chosen. *F-score* is a harmonic mean of precision and recall, balancing the trade-off between false positives and false negatives. It is useful when both precision and recall are important for a specific problem. Considering the high n and low p dimensions of the dataset and the objective of binary classification without the need for interpretability, ensemble models such as RF or GBT might perform best. However, each evaluator's performance will depend on the specific relationships between the features and the target variable in the dataset.

3. Results and Discussion

3.1 Forward/Backward Stepwise Selection/LASSO - White Wine

Interpreting the results of the **LR** for the forward and backward stepwise selection we find that they both produced the same regression, and subsequently, the same results. The **Lasso** regression which included all features produced **slightly worse** results than either of the stepwise selection methods, with an error rate of 0.259184 compared to 0.233. The **most accurate model** is then **forward or backward selection** (see Figure 23).

3.2 Forward/Backward Stepwise Selection/LASSO - Red Wine

Just as for the white wine, the results of the forward and backward stepwise selection methods were the same. The **Lasso** regression proved to be **slightly better** than the stepwise selection methods this time with an error rate of 0.28125 compared to 0.289583. Among the ones tested, the **most accurate model** would be **Lasso** (see Figure 24).

3.3 RFECV - White Wine

In the following paragraph, we will interpret the results of the **RFECV table** for the white wine dataset and **identify the best model for predicting wine quality** (see Figure 25). Both the **ETC and RF appear to be the best-performing models**, with an ROC score of 0.914 and 0.906, an accuracy of 0.837 and 840, a kappa score of 0.619 and 0.628, a precision of 0.848 and 0.854, a recall of 0.920 and 0.916, and an F-score of 0.883 and 0.884 separately. These metrics suggest that the ETC and RF have the **highest overall performance** across the evaluators. They provide a **good balance between precision and recall**, which is crucial for accurately classifying wine quality. The GBT model shows relatively lower performance with an ROC score of 0.856, an accuracy of 0.801, a kappa score of 0.534, a precision of 0.823, a recall of 0.894, and an F-score of 0.857. Despite its lower performance compared to the ETC and RF, it remains a viable option for classification tasks. LR, LDA, and Decision Trees show moderate performance, with ROC scores ranging from 0.923 to 0.749, accuracy scores ranging from 0.777 to 0.767, and kappa scores ranging from 0.498 to 0.444. These models might not be ideal for predicting wine quality in the given dataset, as their performance is not as strong as the ETC or RF. The Ridge Classifier, Linear SVM/SVC, and Perceptron models have NaN values in their ROC scores, which indicates that the evaluation could not be completed successfully. Consequently, it is difficult to assess the performance of these models based on the available metrics. However, their other metrics, such as accuracy, kappa, precision, recall, and F-score, are relatively lower compared to the top-performing models.

3.4 RFECV - Red Wine

In this discussion, we will interpret the results of the RFECV table for the red wine dataset and identify the best model for predicting wine quality (see Figure 26). **The ETC** appears to be the **best-performing model,** with an ROC score of 0.897, an accuracy of 0.816, a kappa score of 0.596, a precision of 0.835, a recall of 0.885, and an F-score of 0.859. These metrics suggest that

the **ETC** has the **highest overall performance** across the evaluators. It provides a good balance between precision and recall, which is crucial for accurately classifying wine quality. The RF model performs similarly to the ETC, with an ROC score of 0.887, an accuracy of 0.807, a kappa score of 0.576, a precision of 0.830, a recall of 0.874, and an F-score of 0.851. Although its performance is slightly lower than the ETC, it is still a strong contender for predicting wine quality. The GBT model shows relatively lower performance with an ROC score of 0.824, an accuracy of 0.752, a kappa score of 0.453, a precision of 0.786, a recall of 0.836, and an F-score of 0.810. Despite its lower performance compared to the ETC and RF, it remains a viable option for classification tasks. LDA and LR show moderate performance, with both ROC scores of 0.792, accuracy scores of 0.719 and 0.721, and kappa scores of 0.384 and 0.385, respectively. These models might not be ideal for predicting wine quality in the given dataset, as their performance is not as strong as the ETC or RF. The Decision Trees model has a missing ROC score, and the other metrics, such as accuracy, kappa, precision, recall, and F-score, are relatively lower compared to the top-performing models. The Ridge Classifier, SVC with a linear kernel, and Linear SVM models have NaN values in their ROC scores, which indicates that the evaluation could not be completed successfully. Consequently, it is difficult to assess the performance of these models based on the available metrics. However, their other metrics, such as accuracy, kappa, precision, recall, and F-score, are relatively lower compared to the top-performing models. The performance metrics of the Perception model are relatively lower, with an accuracy of 0.608 and a kappa score of 0.174. This model is not a suitable choice for predicting wine quality in the given dataset.

3.5 Conclusion

In conclusion, based on the white and red wine datasets, **the ETC appears to be the best model** for predicting wine quality **due to its high performance across various classification metrics**. The RF model also shows promising results and could be considered as an alternative. It is essential to note that the choice of the best model may vary depending on the specific requirements and constraints of a particular application.

4. Appendix

* Note: data preprocessing, data visualization, and RFECV analysis was performed in Python, with approval from Dr. Ning. Forward Stepwise Selection, Backward Stepwise Selection, and Lasso analyses were performed in R.

All Wine

Figure 0. PCA Plot of All Wine Data Set

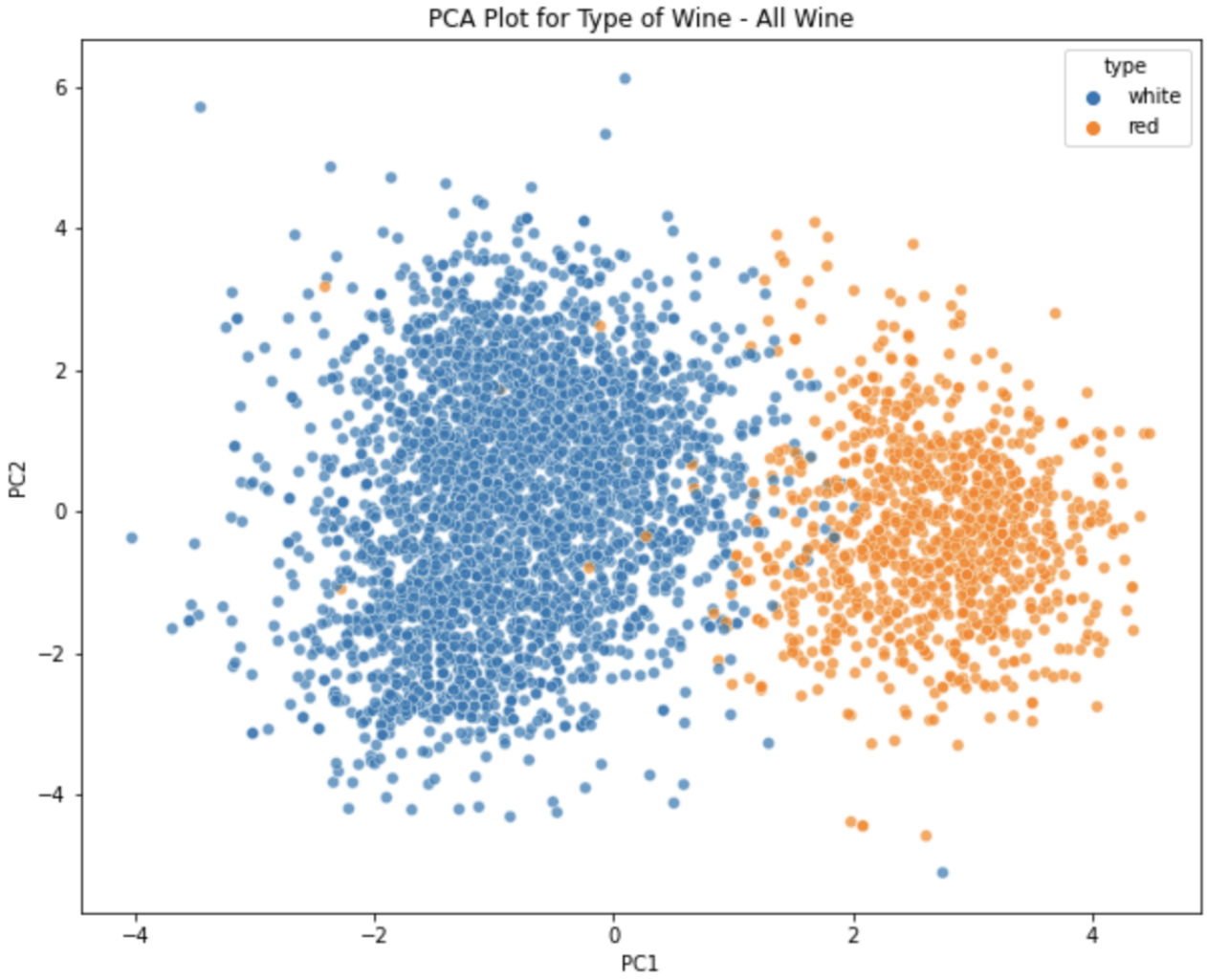


Figure 1: Histogram Plots of All-Wine Dataset—Non-Transformed Data

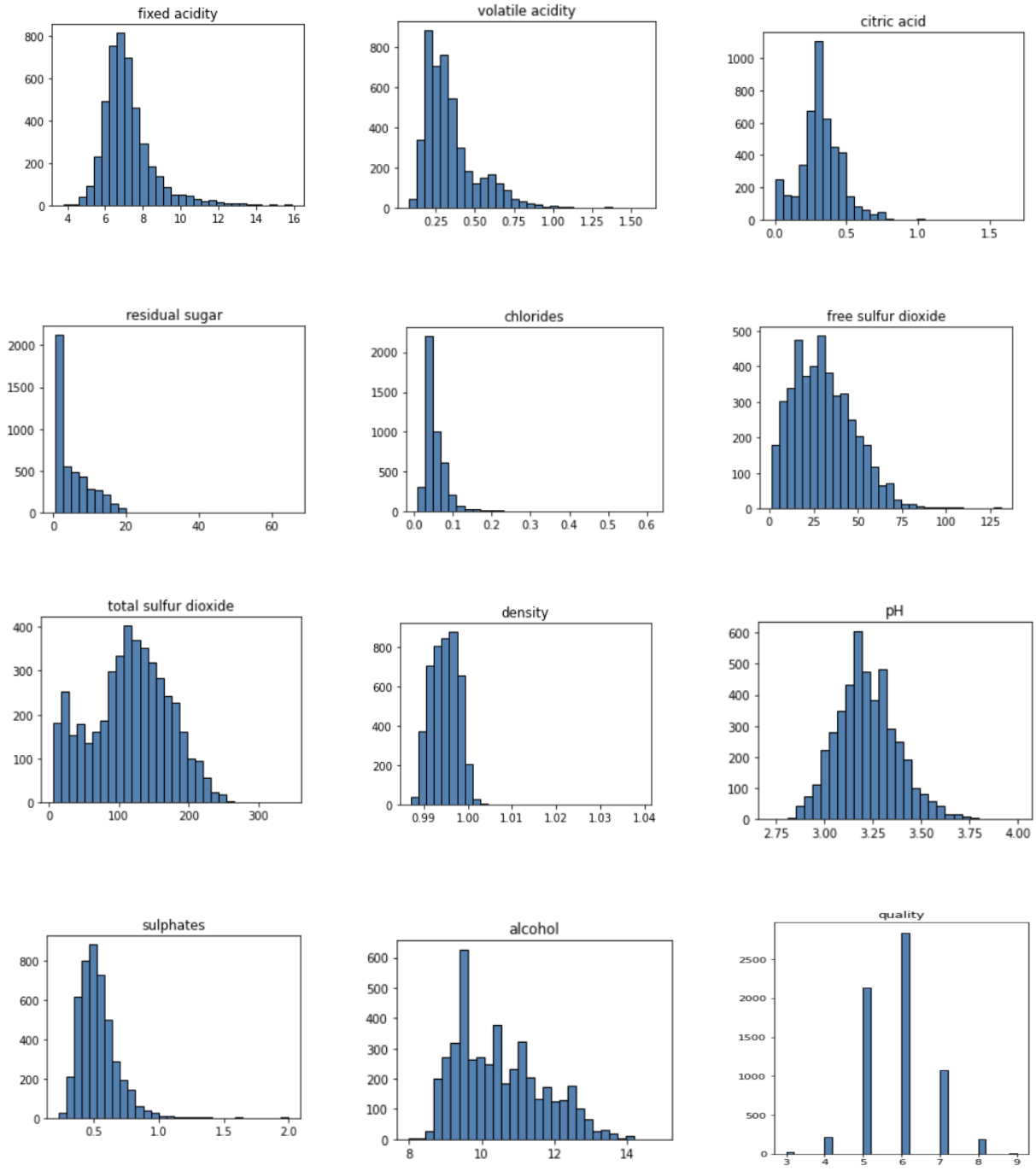


Figure 2: Q-Q Plots of All-Wine Dataset—Non-Transformed Data

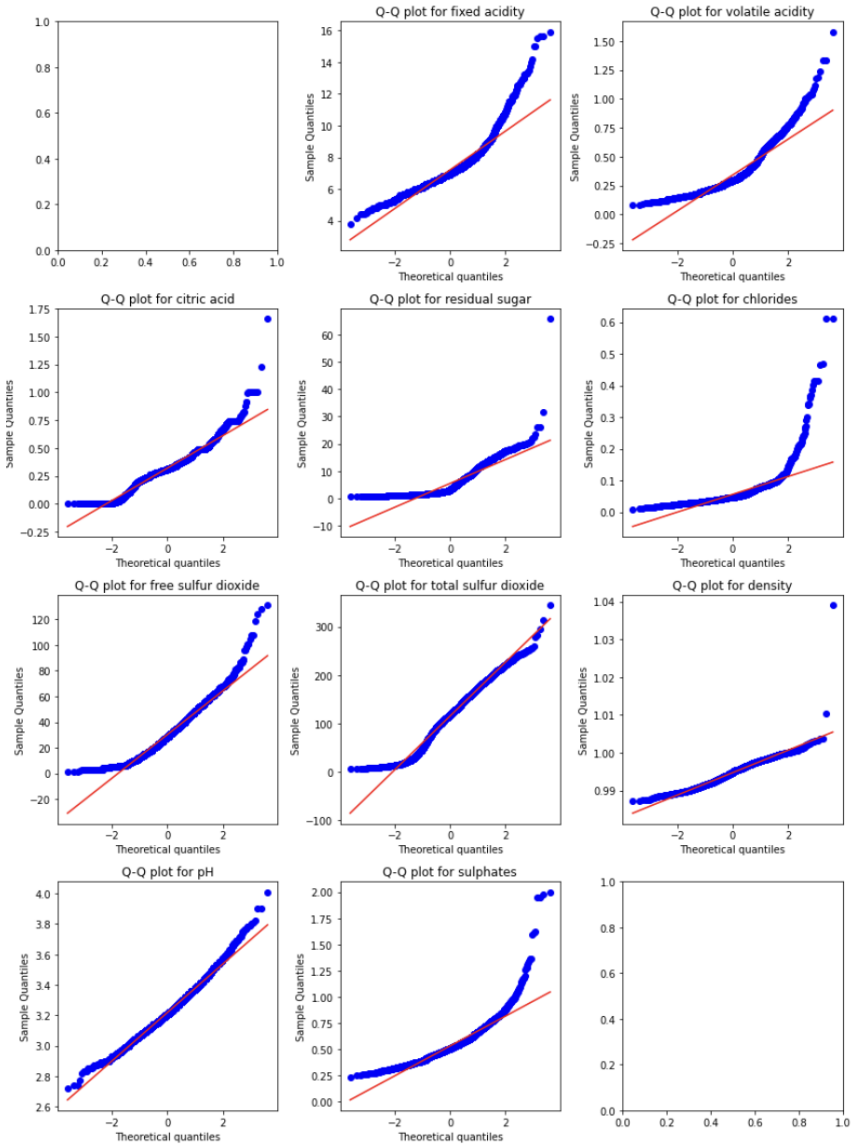


Figure 3: Histogram Plots of All-Wine Dataset—Transformed Data

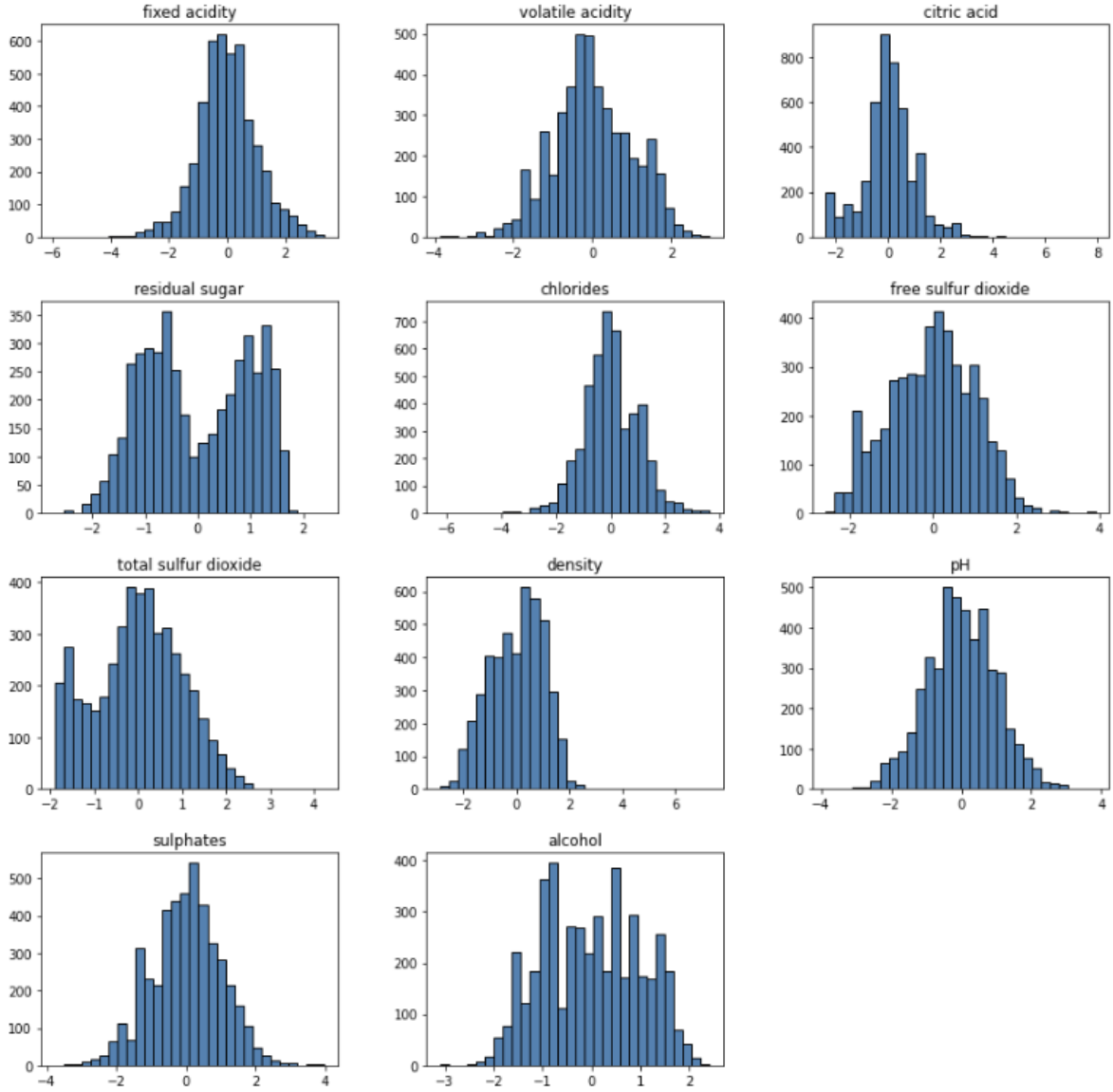


Figure 4: Q-Q Plots of All-Wine Dataset—Transformed Data

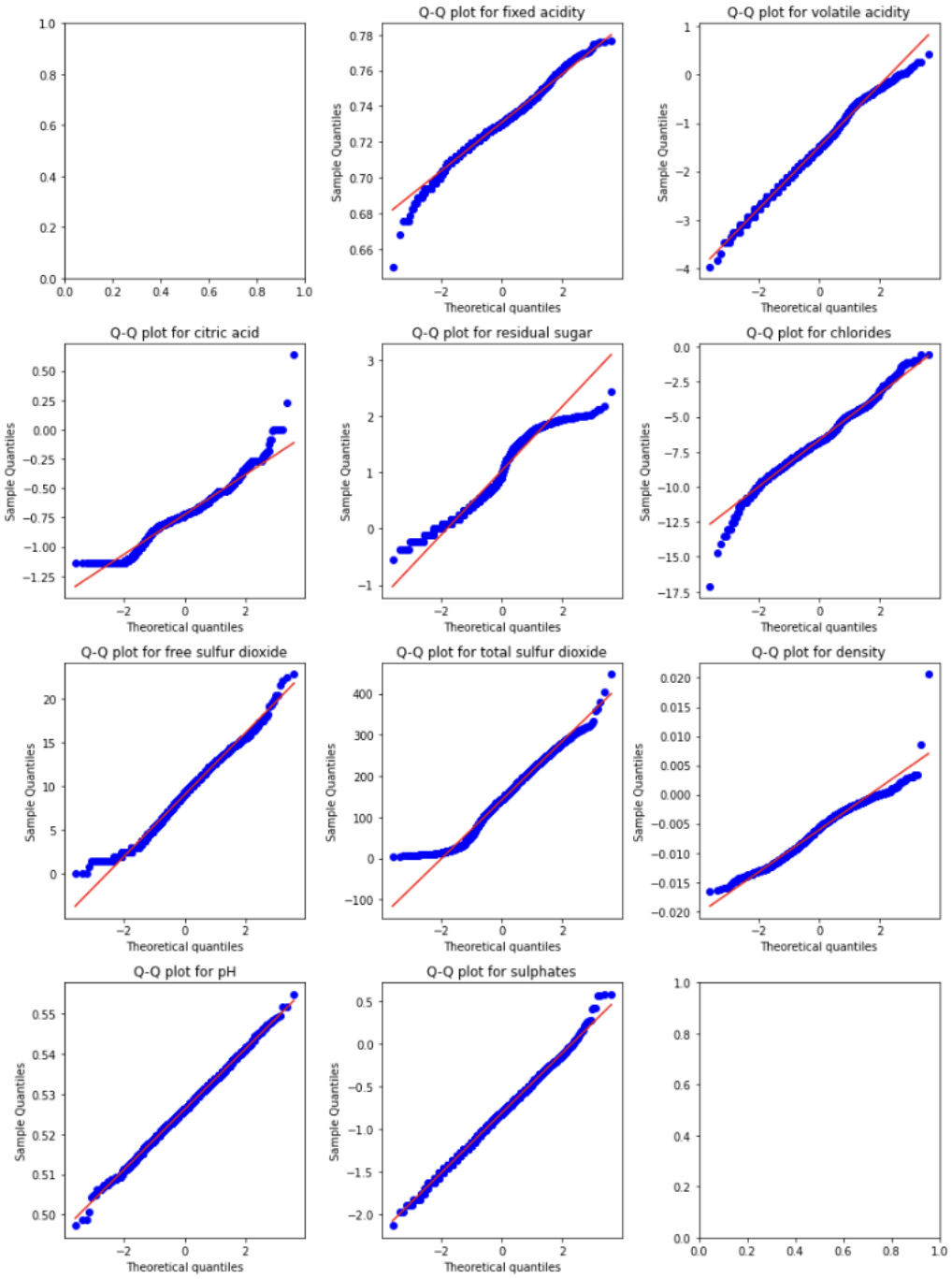


Figure 5: Correlation Matrix Of All-Wine Dataset—Transformed Data

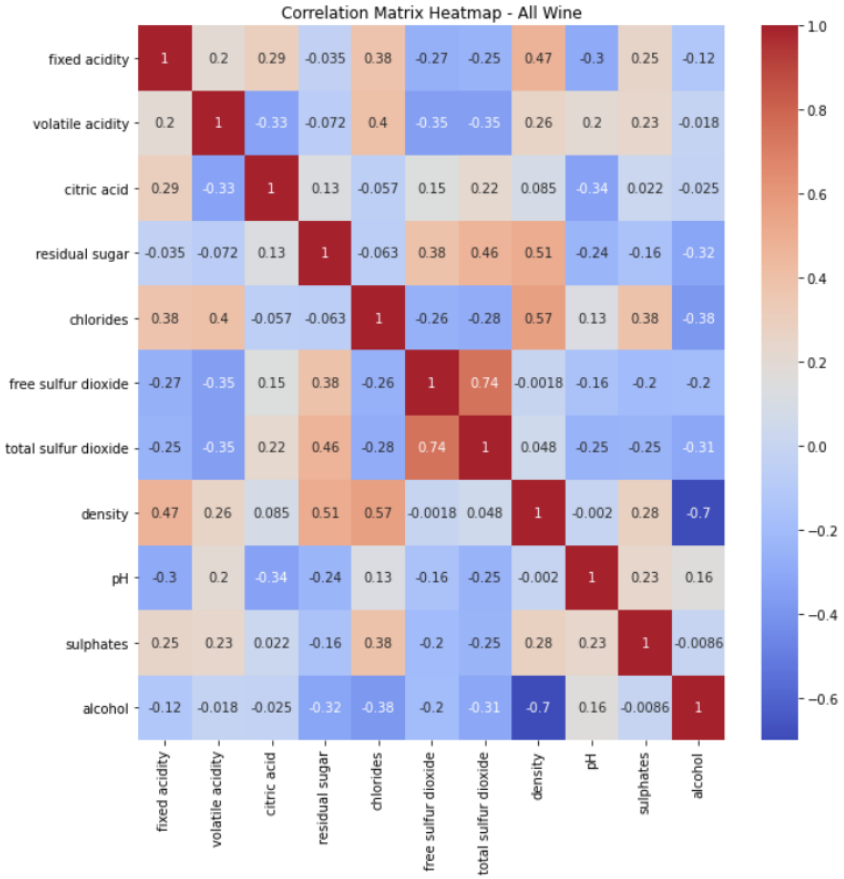


Figure 6: Box Plots of All-Wine Dataset—Transformed Data

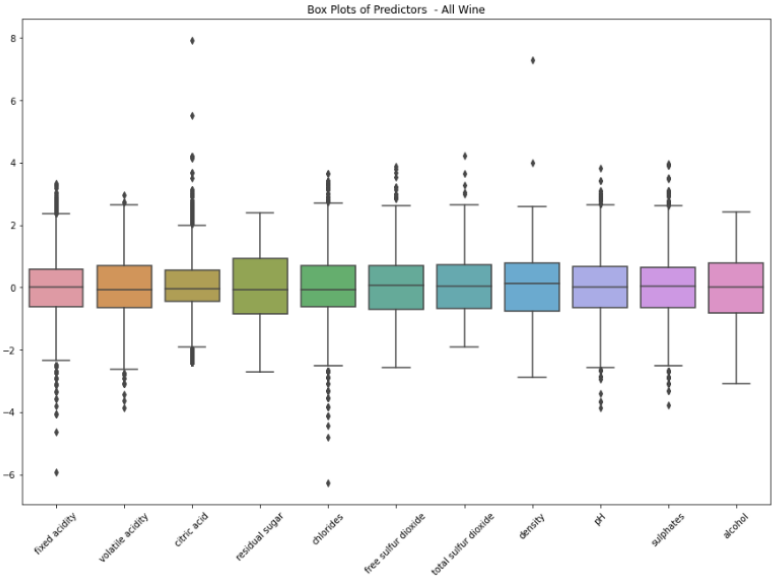
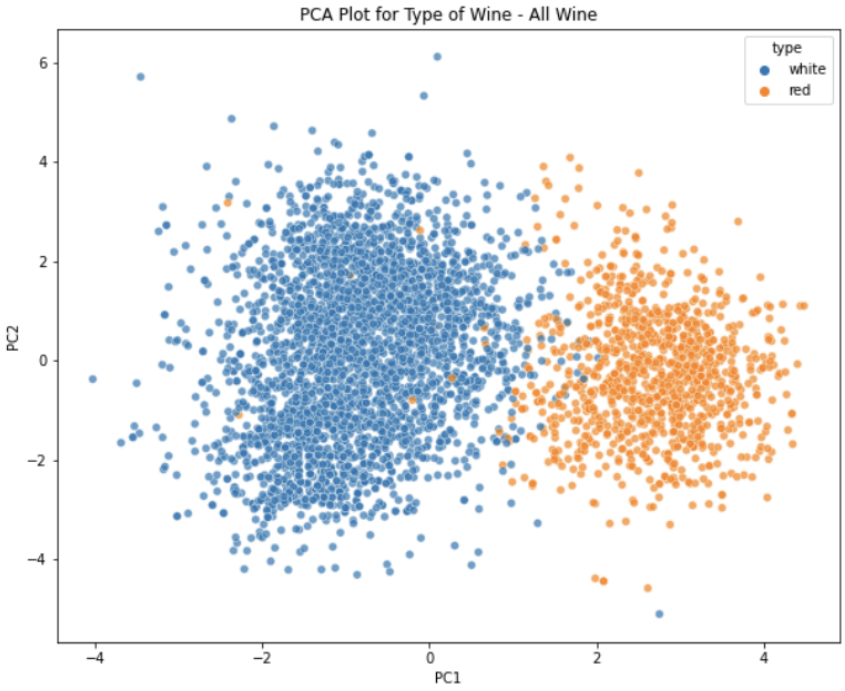


Figure 7: PCA Plot of All-Wine Dataset (colored for quality)—Transformed and Scaled Data

Figure 8: PCA Plot of All-Wine Dataset (colored for wine type)—Transformed and Scaled Data



White Wine

Figure 9: Histogram Plots of White-Wine Dataset—Non-Transformed Data

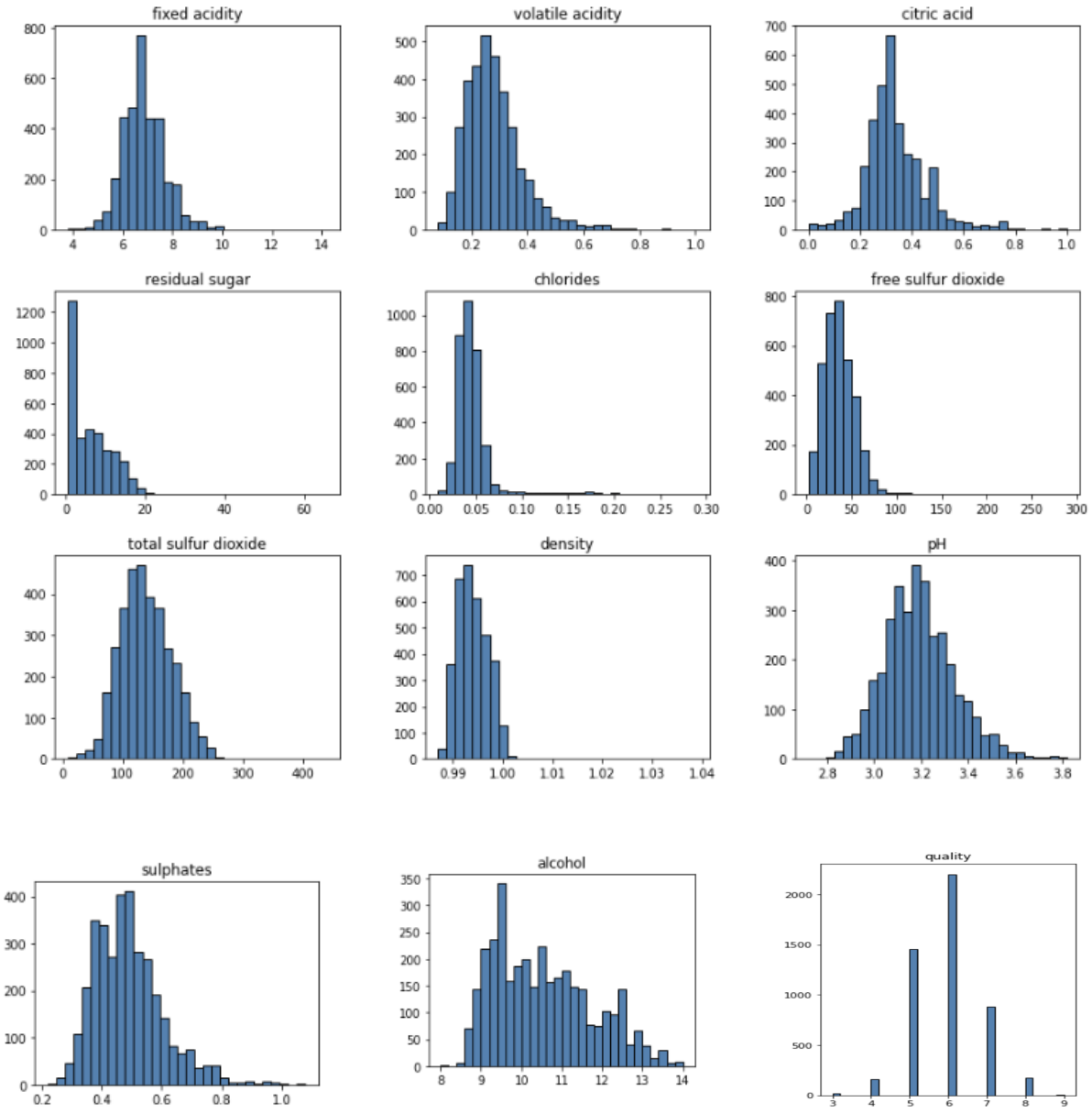


Figure 10: Q-Q Plots of White-Wine Dataset—Non-Transformed Data

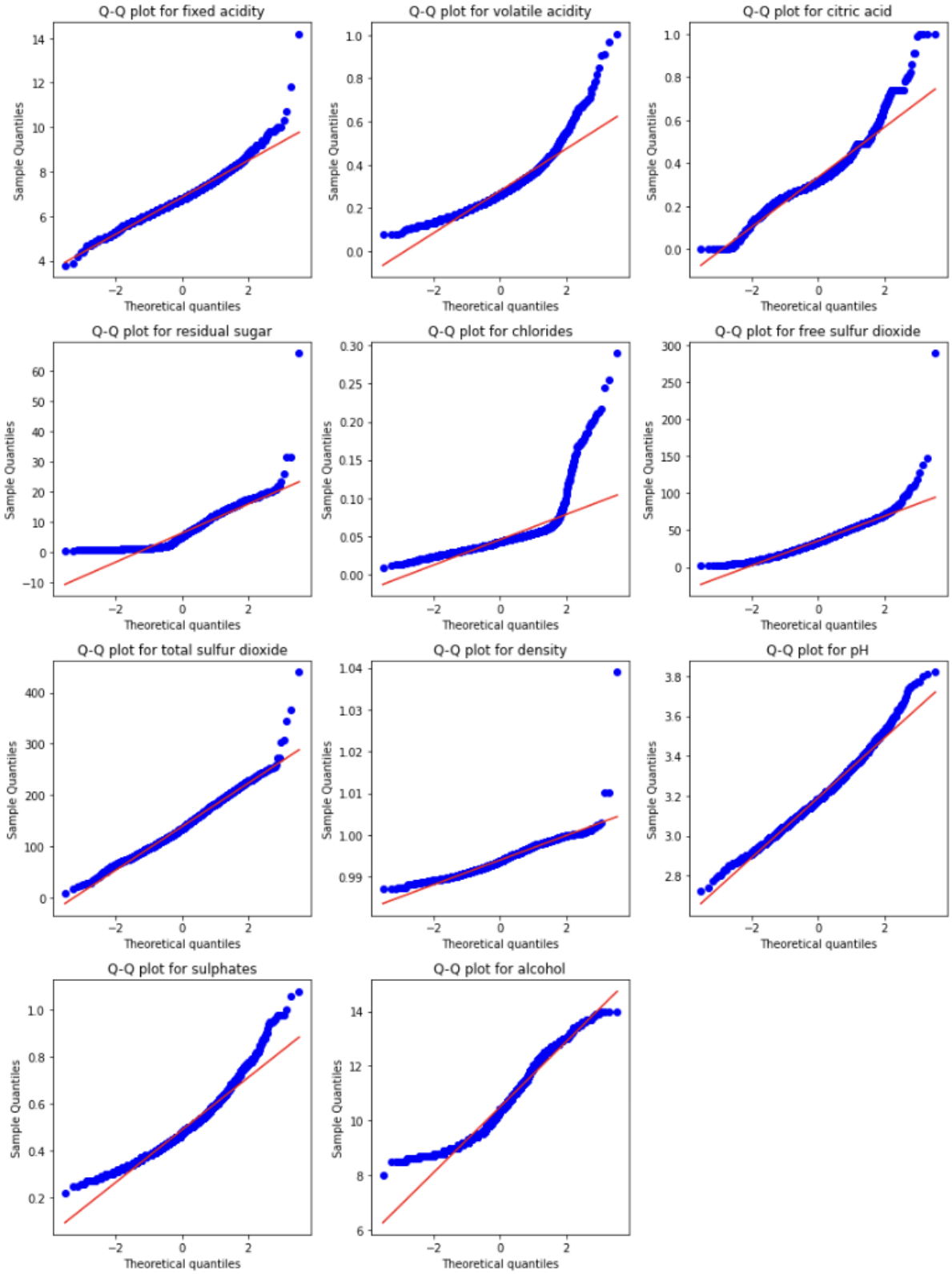


Figure 11: Histogram Plots of White-Wine Dataset—Transformed Data

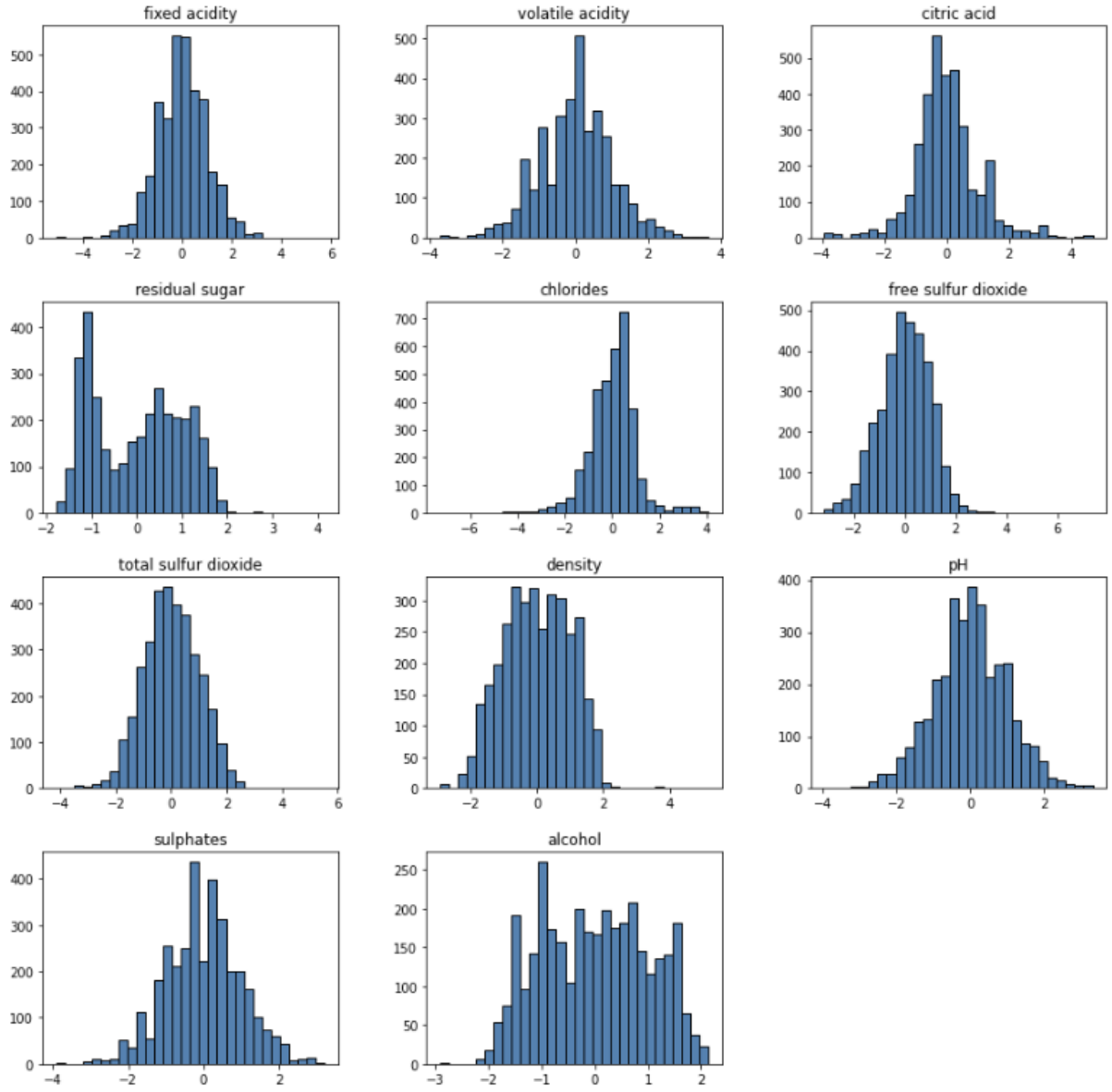


Figure 12: Q-Q Plots of White-Wine Dataset—Transformed Data

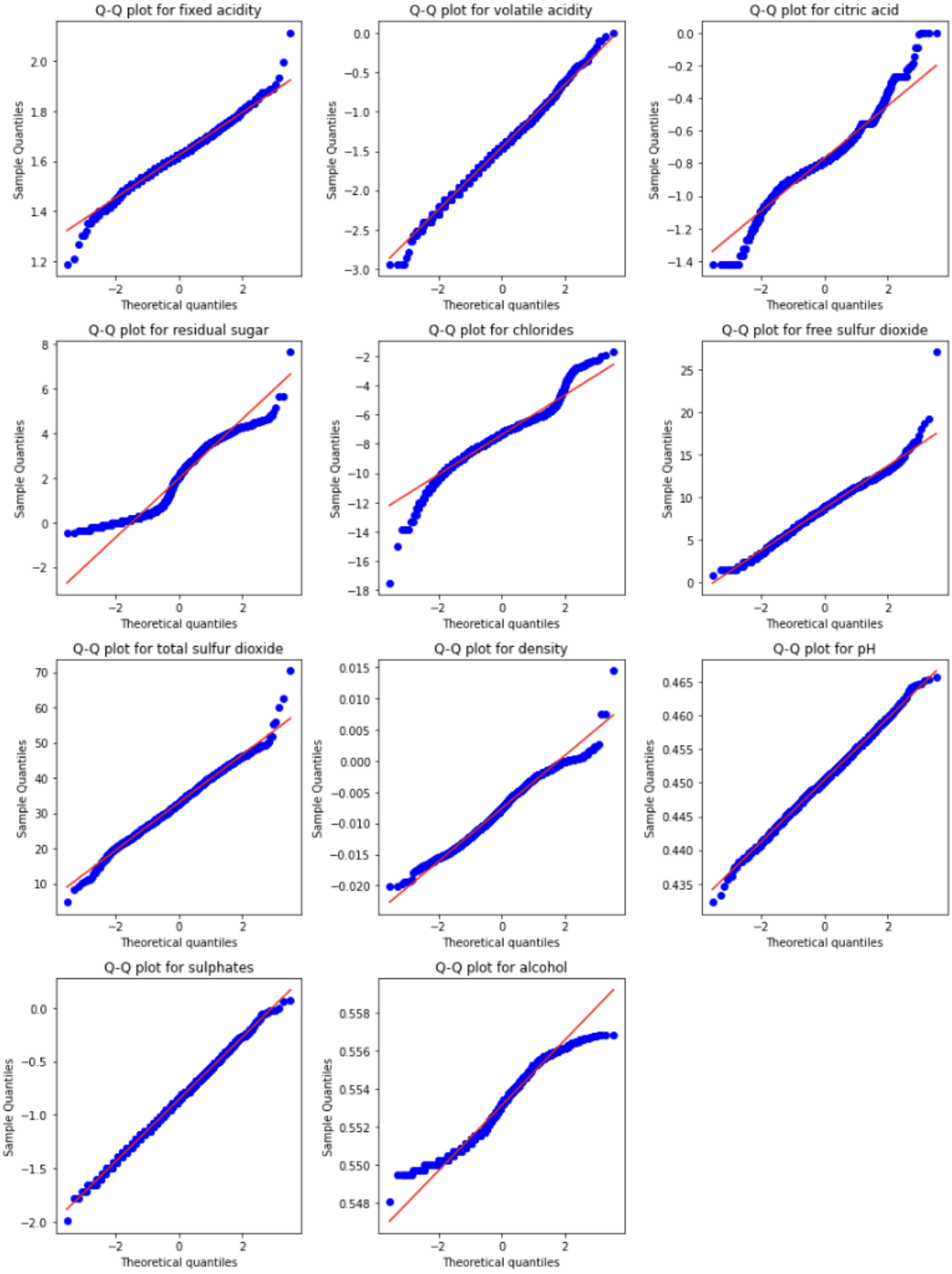


Figure 13: Correlation Matrix Of White-Wine Dataset—Transformed Data

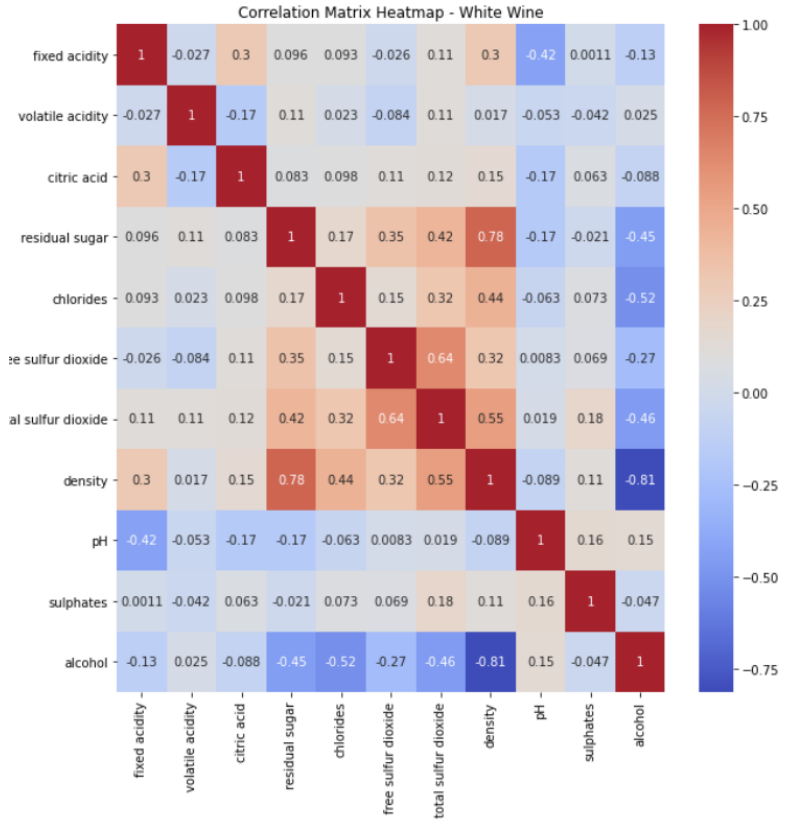


Figure 14: Box Plots of White-Wine Dataset—Transformed Data

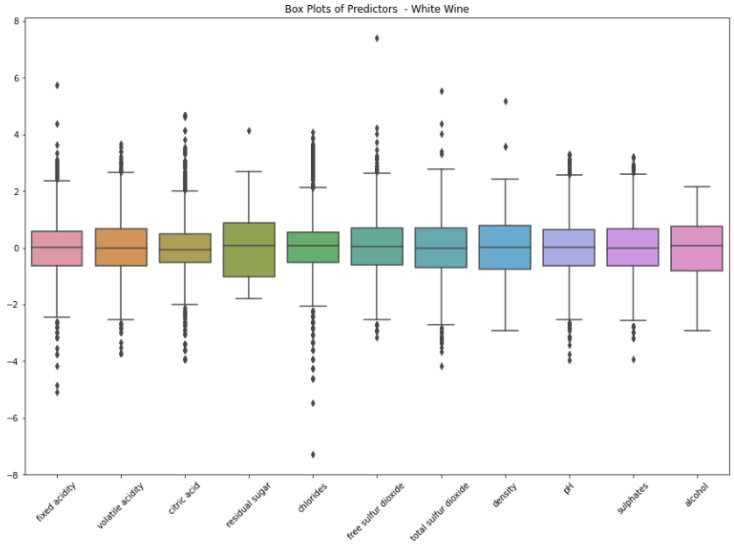


Figure 15: PCA Plot of White-Wine Dataset (colored for quality)—Transformed and Scaled Data

Red Wine

Figure 16: Histogram Plots of Red-Wine Dataset—Non-Transformed Data

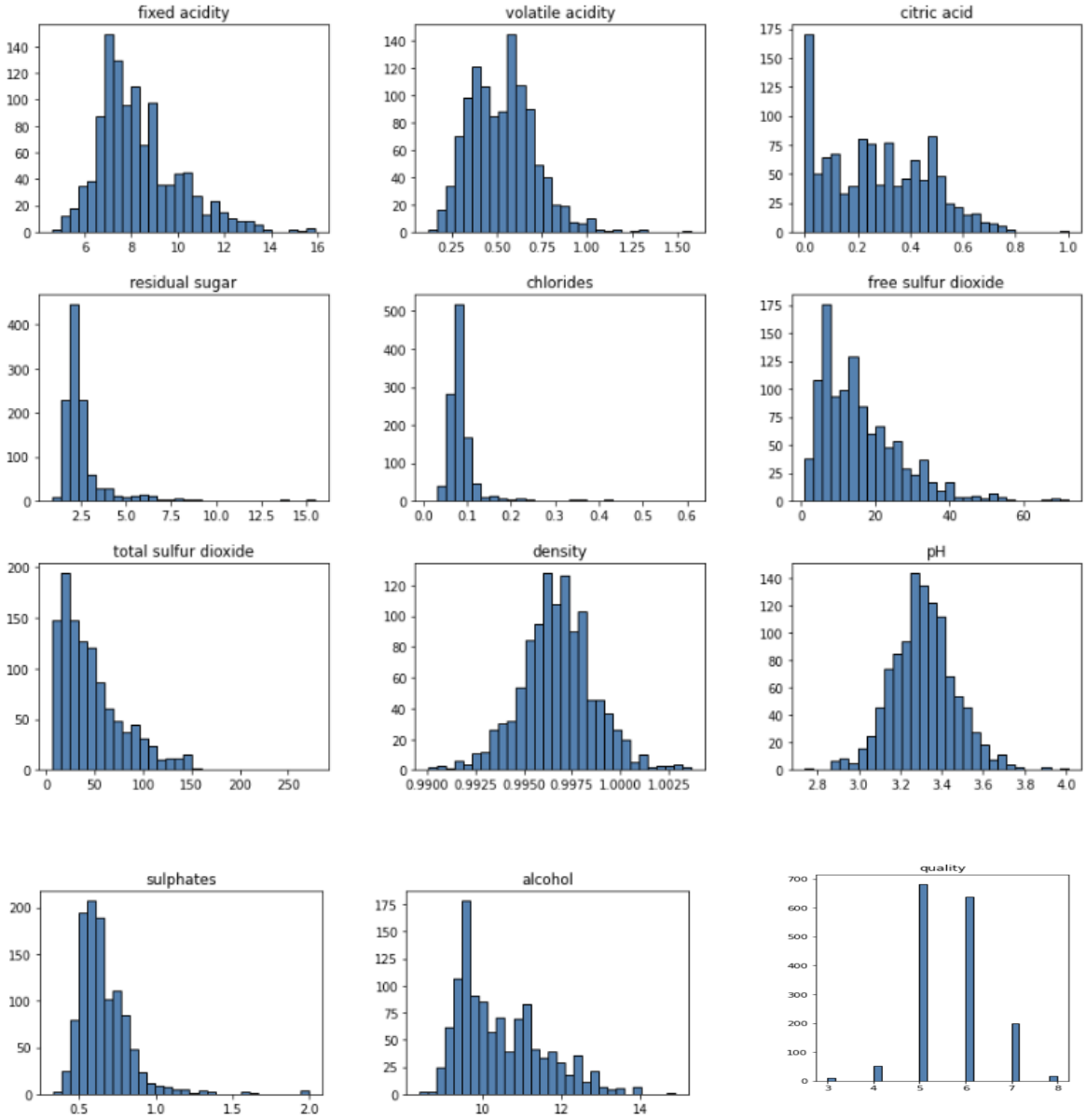


Figure 17: Q-Q Plots of Red-Wine Dataset—Non-Transformed Data

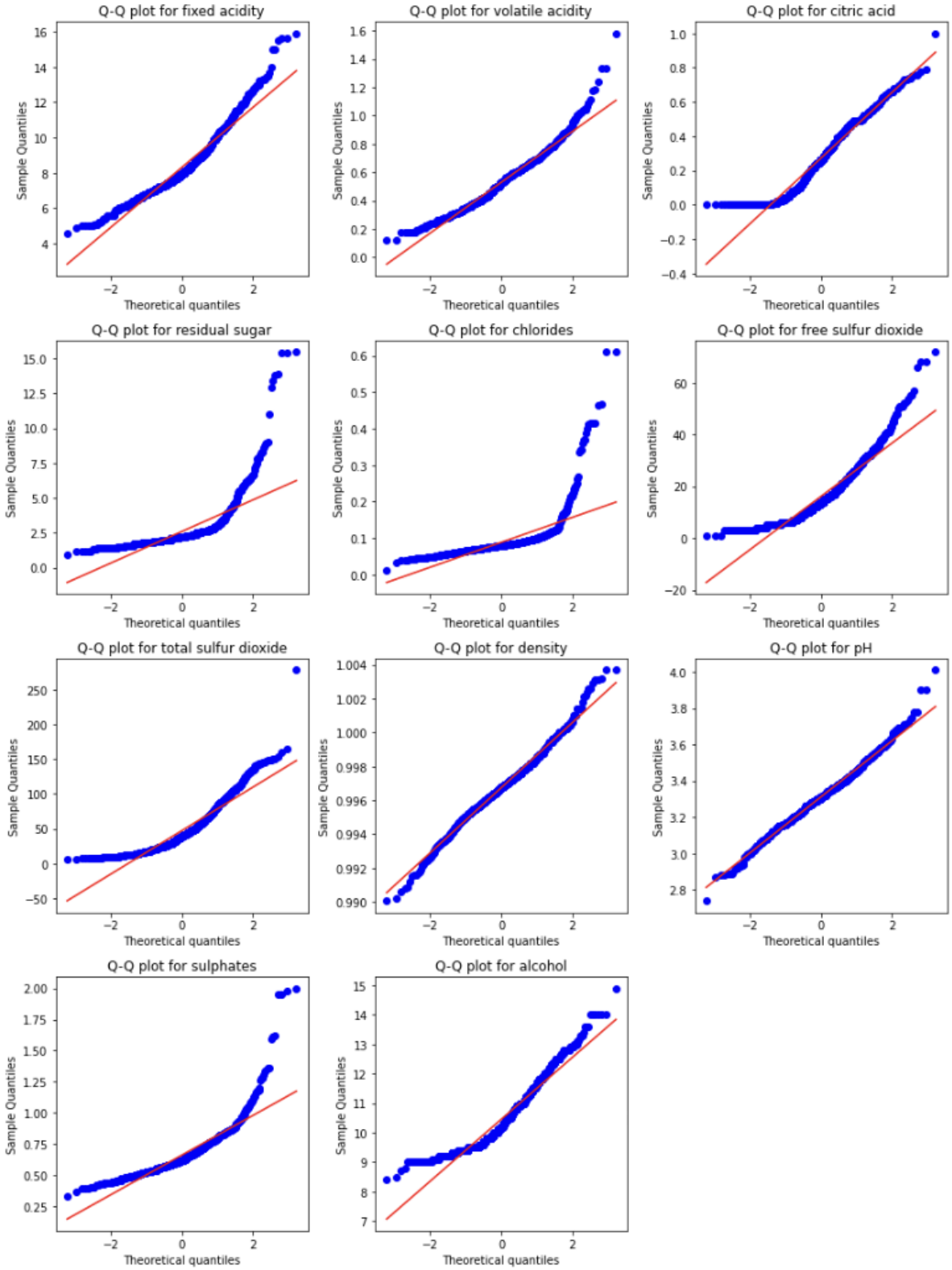


Figure 18: Histogram Plots of Red-Wine Dataset—Transformed Data

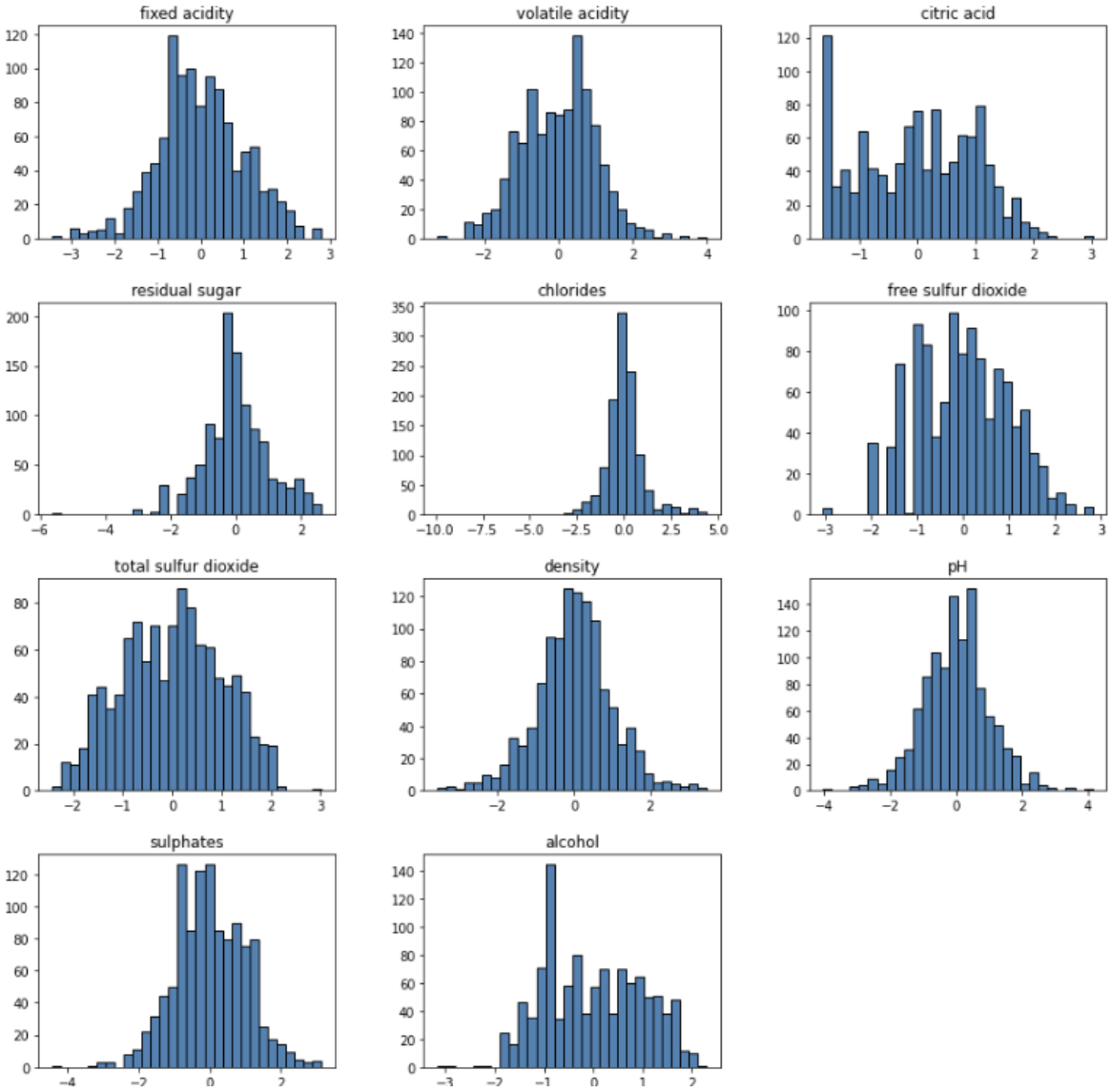


Figure 19: Q-Q Plots of Red-Wine Dataset—Transformed Data

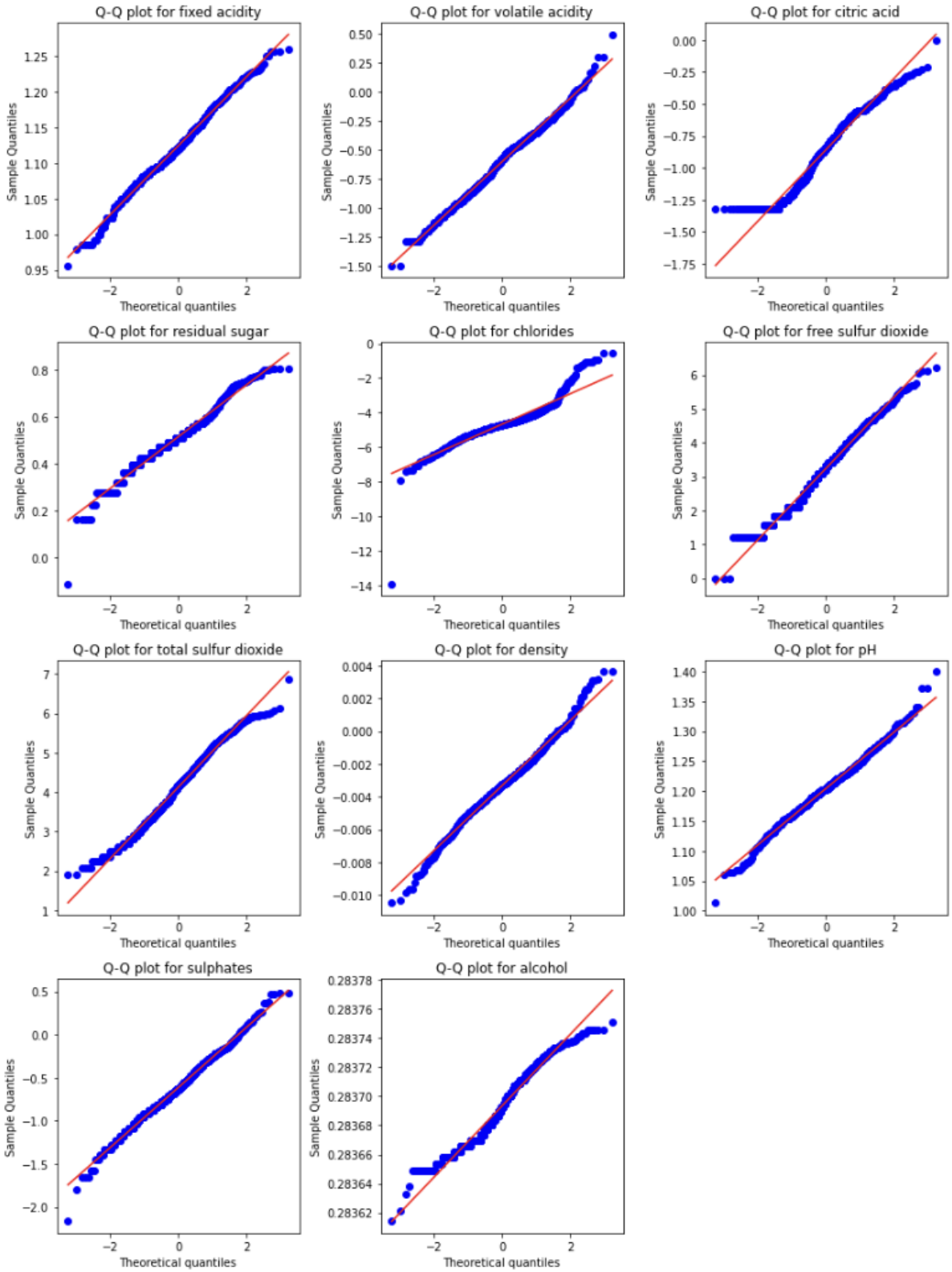


Figure 20: Correlation Matrix Of Red-Wine Dataset—Transformed Data

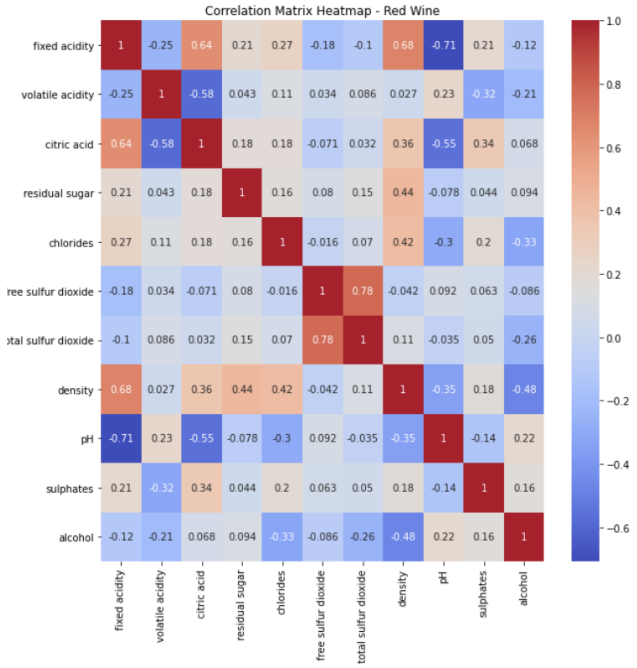


Figure 21: Box Plots of Red-Wine Dataset—Transformed Data

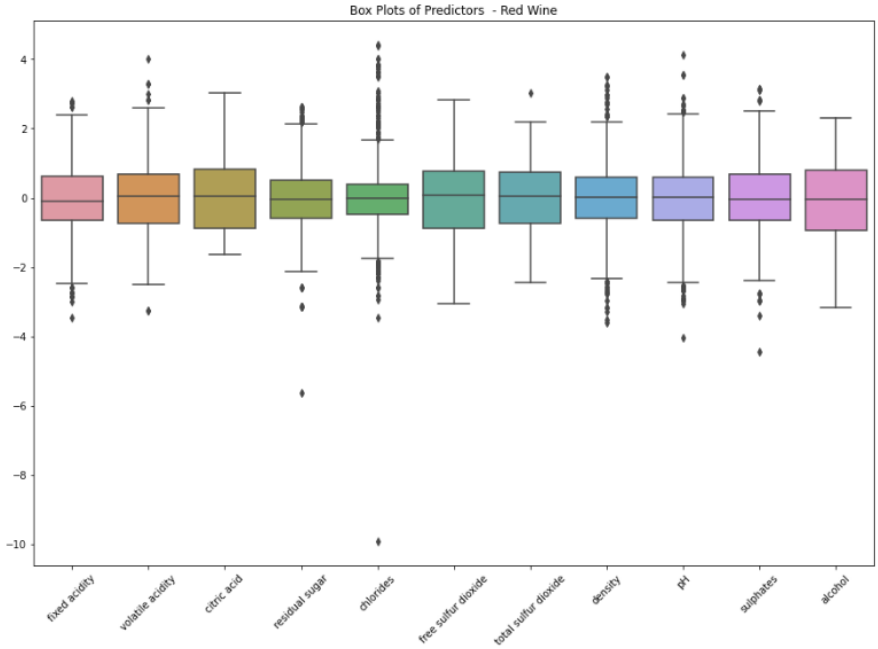


Figure 22: PCA Plot of Red-Wine Dataset (colored for quality)—Transformed and Scaled Data

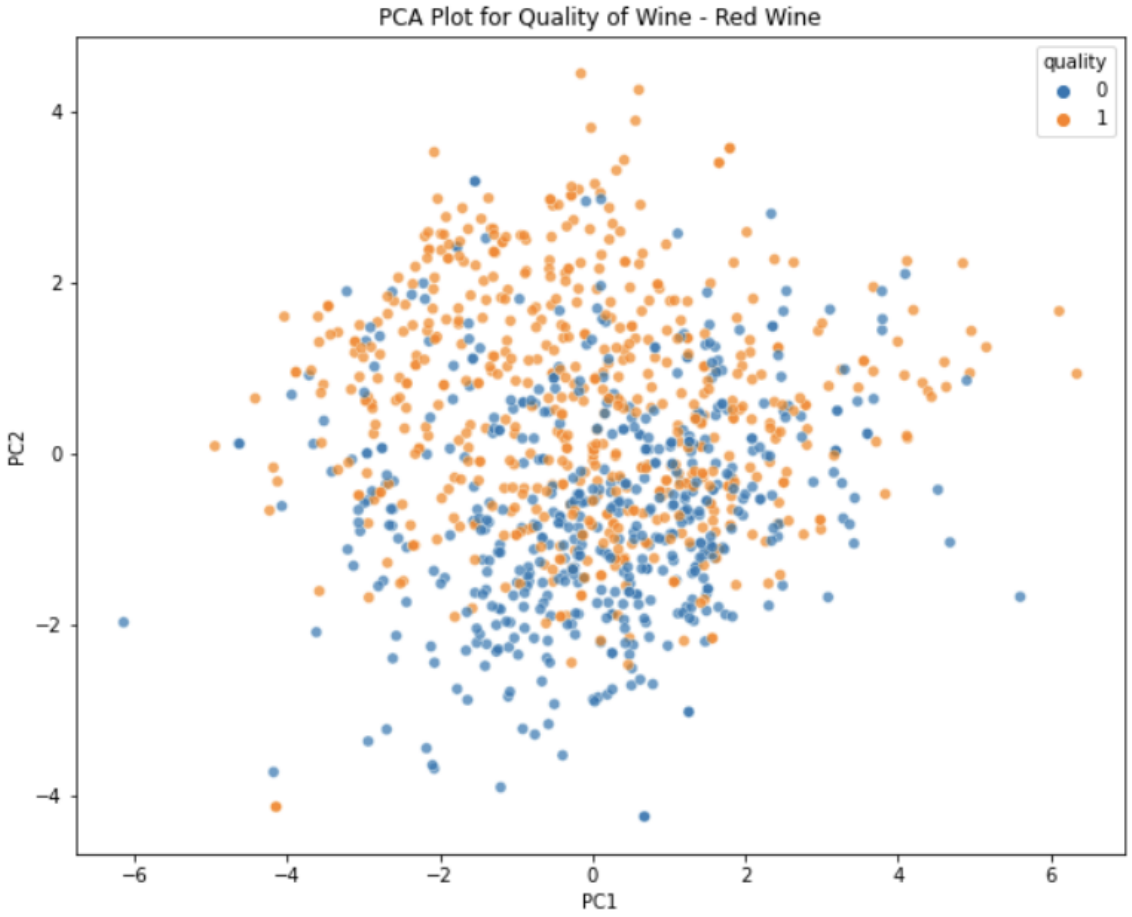


Figure 23: Table of White Wine Dataset Results—Forward Stepwise/Backward Stepwise/Lasso

White Wine

Model	Selected Features	Accuracy
Forward Stepwise Selection: LR	Alcohol,density,fixed acidity, …	0.766667
Backward Stepwise Selection: LR	Alcohol,density,fixed acidity,...	0.766667
Lasso	All features included	0.740816

Figure 24: Table of Red Wine Dataset Results—Forward Stepwise/Backward Stepwise/Lasso

Red Wine

Model	Selected Features	Accuracy
Forward Stepwise Selection: LR	Alcohol,density,fixed acidity, …	0.71042
Backward Stepwise Selection: LR	Alcohol,density,fixed acidity,...	0.71042
Lasso	All features included	0.71875

Figure 25: Table of White Wine Dataset Results—RFECV

Evaluator	Selected Features	ROC	Accuracy	Kappa	Precision	Recall	F-score
Extra Trees Classifier	fixed acidity, volatile acidity, citric acid, ...	0.914230	0.837415	0.618954	0.848256	0.920245	0.882786
Random Forest	fixed acidity, volatile acidity, citric acid, ...	0.906495	0.840136	0.627686	0.854147	0.916155	0.884065
Gradient Boosting Trees	fixed acidity, volatile acidity, citric acid, ...	0.856480	0.801361	0.534208	0.822976	0.893661	0.856863
LDA	fixed acidity, volatile acidity, residual suga...	0.822693	0.767347	0.444489	0.790146	0.885481	0.835101
Logistic Regression	fixed acidity, volatile acidity, residual suga...	0.822053	0.768027	0.445219	0.789809	0.887526	0.835821
Decision Trees	fixed acidity, volatile acidity, citric acid, ...	0.748984	0.776871	0.498471	0.831633	0.833333	0.832482
Ridge Classifier	fixed acidity, volatile acidity, residual suga...	NaN	0.768027	0.440403	0.785650	0.895706	0.837076
SVC with linear kernel	fixed acidity, volatile acidity, residual suga...	NaN	0.768707	0.447143	0.790528	0.887526	0.836224
Perceptron	fixed acidity, residual sugar, chlorides, free...	NaN	0.641497	0.076014	0.682887	0.860941	0.761646
Linear SVM	fixed acidity, volatile acidity, residual suga...	NaN	0.770068	0.447425	0.788809	0.893661	0.837967

Figure 26: Table of Red Wine Dataset Results—RFECV

Evaluator	Selected Features	ROC	Accuracy	Kappa	Precision	Recall	F-score
Extra Trees Classifier	fixed acidity, volatile acidity, citric acid, ...	0.897418	0.816410	0.595906	0.834989	0.885020	0.859277
Random Forest	type, fixed acidity, volatile acidity, citric ...	0.886946	0.806667	0.575610	0.830000	0.873684	0.851282
Gradient Boosting Trees	fixed acidity, volatile acidity, citric acid, ...	0.823972	0.751795	0.453025	0.785986	0.835628	0.810047
Logistic Regression	type, fixed acidity, volatile acidity, residua...	0.791724	0.720513	0.385015	0.763761	0.808907	0.785686
LDA	type, fixed acidity, volatile acidity, residua...	0.791577	0.718974	0.384408	0.765661	0.801619	0.783228
Decision Trees	fixed acidity, volatile acidity, chlorides, fr...	0.728193	0.744103	0.453051	0.804132	0.787854	0.795910
Ridge Classifier	type, fixed acidity, volatile acidity, residua...	NaN	0.716410	0.376738	0.761503	0.804049	0.782198
SVC with linear kernel	volatile acidity, residual sugar, free sulfur ...	NaN	0.721026	0.388901	0.767208	0.803239	0.784810
Perceptron	fixed acidity, volatile acidity, citric acid, ...	NaN	0.608205	0.173935	0.702494	0.661538	0.681401
Linear SVM	type, fixed acidity, volatile acidity, residua...	NaN	0.718974	0.383300	0.764434	0.804049	0.783741

In [ ]:
```
from scipy.stats import shapiro
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis, Quadratic
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.feature_selection import RFECV
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LogisticRegression, RidgeClassifier, Lasso, El
from sklearn.metrics import accuracy_score, cohen_kappa_score, precision_score,
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.metrics import roc_auc_score, accuracy_score, cohen_kappa_score
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.model_selection import train_test_split, StratifiedKFold, cross_va
from sklearn.naive_bayes import GaussianNB
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import FunctionTransformer, StandardScaler
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.svm import SVC, LinearSVC
from scipy import stats
from scipy.stats import boxcox, boxcox_normmax,probplot
import pickle
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import random
import seaborn as sns
```

In [2]:
```
# Normality & Log Transform
def log_transform_if_not_normal(X_train, X_validation):
    for col in X_train.columns:
        if col in ["type"]:
            print(col)
            continue
        lambda_opt = boxcox_normmax(X_train[col] + 1e-5)
        # Apply Box-Cox transformation using the optimal lambda to training and
        X_train[col] = boxcox(X_train[col] + 1e-5, lmbda=lambda_opt)
        X_validation[col] = boxcox(X_validation[col] + 1e-5, lmbda=lambda_opt)


## Must remember later to log transofrm validations et columns
```

In [3]:
```
#### Loading Dataset ####

random.seed(10)

# Load Dataset
all_wine = pd.read_csv("/hpc/users/parkr02/minerva_jobs/jupyter_jobs/STSCI/wine

red_wine = all_wine[all_wine['type'] == 'red'].copy().drop(['type'],axis=1)
white_wine = all_wine[all_wine['type'] == 'white'].copy().drop(['type'],axis=1)
```

In [4]:
```
def histogram(X, X_columns, boolean):
    if boolean:
        X = pd.DataFrame(X,columns = X_columns.columns)
        X.hist(figsize=(15, 15), bins=30, grid=False, color="steelblue", edgecc
        plt.show()
    else:
        X = pd.DataFrame(X['quality'], columns = ['quality'])
        X.hist(figsize=(4, 6), bins=30, grid=False, color="steelblue", edgecolc
        plt.show()
```

In [5]:
```
def qqplots(X_train):
    # (6) Q-Q plots
    num_cols = 3
    num_rows = (len(X_train.columns) + num_cols - 1) // num_cols
    fig, axs = plt.subplots(num_rows, num_cols, figsize=(12, 4*num_rows))
    for idx, col in enumerate(X_train.columns):
        if X_train[col].dtype != "object":
            i, j = divmod(idx, num_cols)
            probplot(X_train[col], dist="norm", plot=axs[i][j])
            axs[i][j].set_title("Q-Q plot for " + col)
            axs[i][j].set_ylabel("Sample Quantiles")
        if idx == (num_rows * num_cols) - 2: # -1 for zero-based index and -1 f
            break
    if len(X_train.columns) % 3 != 0: # add empty plot if there are fewer than
        i, j = divmod(len(X_train.columns) - 1, num_cols)
        axs[i][j+1].axis("off")
    plt.tight_layout()
    plt.show()
```

In [6]:
```
def plot_dataset(data1, title):
    random.seed(10)
    data = data1.copy()
    # Binary Code Qualitative Variables (Good Quality = 1 ; Bad Quality = 0)
    data['quality'] = (data['quality'] > 5).astype(int)
    # Test Train Split
    X = data.copy().drop(columns=['quality'])
    y = data.copy()['quality']
    if "All" in title:
        X_train, X_validation, y_train, y_validation = train_test_split(X, y, s
    else:
        X_train, X_validation, y_train, y_validation = train_test_split(X, y, s

    # BoxCox Transformation #
    # Note: non-normal in training set
    print("\n")
    print("Pre Box Cox Transformation & Standard Scaling")

    histogram(X_train.copy(),X_train, True)

    histogram(data1.copy(),data1.copy(), False)

    qqplots(X_train.copy())

    log_transform_if_not_normal(X_train, X_validation)

    # Standard Scaling for PCA & BoxPlots
```

```
    scaler = StandardScaler()
    if "type" in X_train.columns:
        X_train_drop = X_train.copy().drop("type", axis=1)
    else:
        X_train_drop = X_train.copy()
    X_train_scaled = scaler.fit_transform(X_train_drop)
    print("\n")
    print("Post Log Transformation & Standard Scaling")
    # (1) Correlation Matrix on Dataset
    corr_matrix = X_train.corr()
    plt.figure(figsize=(10, 10))
    sns.heatmap(corr_matrix, cmap='coolwarm', annot=True)
    plt.title('Correlation Matrix Heatmap - ' + title)
    plt.show()

    # (2) Box Plots
    plt.figure(figsize=(15, 10))
    sns.boxplot(data=pd.DataFrame(X_train_scaled,columns = X_train_drop.columns
    plt.xticks(rotation=45)
    plt.title("Box Plots of Predictors  - " + title)
    plt.show()

    # (3) PCA Plot


    pca = PCA(n_components=2)
    pca_result = pca.fit_transform(X_train_scaled)
    pca_df = pd.DataFrame(pca_result, columns=['PC1', 'PC2'])
    pca_df['quality'] = y_train.values

    plt.figure(figsize=(10, 8))
    sns.scatterplot(data=pca_df, x='PC1', y='PC2', hue='quality', alpha=0.7)
    plt.title('PCA Plot for Quality of Wine - ' + title)
    plt.show()

    # (4) Repeat PCA plot but color for Wine Type (if "All" in Title)
    if "All" in title and "type" in X_train.columns:
        pca_df = pd.DataFrame(pca_result, columns=['PC1', 'PC2'])
        pca_df['type'] = X_train['type'].values

        plt.figure(figsize=(10, 8))
        sns.scatterplot(data=pca_df, x='PC1', y='PC2', hue='type', alpha=0.7)
        plt.title('PCA Plot for Type of Wine - All Wine')
        plt.show()

    # (5) Histograms

    histogram(X_train_scaled,X_train_drop, True)

    # (6) Q-Q plots
    qqplots(X_train)
    return X_train, y_train, X_validation, y_validation
```

In [ ]:
```
AW = plot_dataset(all_wine, "All Wine")
```

In [ ]:
```
WW = plot_dataset(white_wine, "White Wine")
```

In [ ]:
```
RW = plot_dataset(red_wine, "Red Wine")
```

In [10]:
```
# Model Selection
def rfecv_all(Split, n_splits=10, n_repeats=5):
    if 'type' in Split[0].columns:
        print("worked")
        Split[0]['type'] = (Split[0]['type'] == 'red').astype(int)
        Split[2]['type'] = (Split[2]['type'] == 'red').astype(int)

    X = Split[0]
    y = Split[1]
    models = [
        ('Ridge Classifier', RidgeClassifier()),  # Added
        ('Logistic Regression', LogisticRegression(solver='liblinear')),
        ('LDA', LinearDiscriminantAnalysis()),
        ('SVC with linear kernel', SVC(kernel='linear')),  # Added
        ('Decision Trees', DecisionTreeClassifier()),  # Added
        ('Random Forest', RandomForestClassifier(n_estimators=100, random_state
        ('Gradient Boosting Trees', GradientBoostingClassifier()),  # Added
        ('Extra Trees Classifier', ExtraTreesClassifier()),  # Added
        ('Perceptron', Perceptron()),  # Added
        ('Linear SVM', LinearSVC())
    ]
    results = []
    fitted_models = {}

    for name, model in models:
        print(name)
        cv = RepeatedStratifiedKFold(n_splits=n_splits, n_repeats=n_repeats, ra

        feature_selector = RFECV(model, step=1, cv=cv, scoring='roc_auc', verbo
        pipe = Pipeline([
            ('scaler', StandardScaler()),
            ('feature_selector', feature_selector),
            ('model', model)
        ])

        cv_scores = cross_val_score(pipe, X, y, cv=cv, scoring='roc_auc')
        pipe.fit(X, y)

        y_pred = pipe.predict(X)
        selected_features = X.columns[pipe.named_steps['feature_selector'].get_
        roc = cv_scores.mean()
        accuracy = accuracy_score(y, y_pred)
        kappa = cohen_kappa_score(y, y_pred)
        precision = precision_score(y, y_pred)
        recall = recall_score(y, y_pred)
        f_score = f1_score(y, y_pred)

        results.append({
            'Evaluator': name,
            'Selected Features': ', '.join(selected_features),
            'ROC': roc,
            'Accuracy': accuracy,
            'Kappa': kappa,
            'Precision': precision,
```

```
            'Recall': recall,
            'F-score': f_score
        })

        # Store the fitted models
        fitted_models[name] = pipe

        # Plot for forward selection with error bars
        fig, ax = plt.subplots()

        avg_scores = np.mean(feature_selector.grid_scores_,axis=1)
        std_devs = feature_selector.grid_scores_.std(axis=1)

        ax.errorbar(range(1, len(avg_scores) + 1), avg_scores, yerr=std_devs, f
        ax.set_xlabel('Number of Features')
        ax.set_ylabel('Cross-Validation Score')
        ax.set_title(f'RFECV: {name}')
        plt.show()

    results_df = pd.DataFrame(results)
    results_df = results_df.sort_values(by='ROC', ascending=False).reset_index(
    return results_df, fitted_models, Split # Split is output from prior functi
```

In [11]:
```
file_name = "AllWine.pkl"
with open(file_name, 'wb') as f:
    pickle.dump(AW,f)
```

In [ ]:
```
AW_Models = rfecv_all(AW)
```

In [ ]:
```
file_name = "AllWineModels.pkl"
with open(file_name, 'wb') as f:
    pickle.dump(AW_Models,f)
```

In [ ]:
```
WW_Models = rfecv_all(WW)
```

In [ ]:
```
file_name = "WhiteWineModels.pkl"
with open(file_name, 'wb') as f:
    pickle.dump(WW_Models,f)
```

In [ ]:
```
RW_Models = rfecv_all(RW)
```

In [ ]:
```
file_name = "RedWineModels.pkl"
with open(file_name, 'wb') as f:
    pickle.dump(RW_Models,f)
```

In [ ]:
```
import os
directory_in_str = '/hpc/users/parkr02/minerva_jobs/jupyter_jobs/STSCI/'
directory = os.fsencode(directory_in_str)

Features = []
for file in os.listdir(directory):
    filename = os.fsdecode(file)
    if filename.endswith(".pkl"):
        with open(filename,'rb') as f:
            data = pickle.load(f)
```

```
            Features.append(data)
    else:
        continue
```

In [12]:
```
from sklearn.metrics import accuracy_score, cohen_kappa_score, roc_auc_score, p
# Validation Set
def predict_and_evaluate(Models):
    X_val = Models[2][2]
    y_val = Models[2][3]
    fitted_models = Models[1]

    results = []

    for name, model in fitted_models.items():
        y_pred = model.predict(X_val)
        try:
            y_proba = model.predict_proba(X_val)[:, 1]
            roc = roc_auc_score(y_val, y_proba)
        except:
            roc = None
        selected_features = X_val.columns[model.named_steps['feature_selector']

        accuracy = accuracy_score(y_val, y_pred)
        kappa = cohen_kappa_score(y_val, y_pred)
        precision = precision_score(y_val, y_pred)
        recall = recall_score(y_val, y_pred)
        f_score = f1_score(y_val, y_pred)

        results.append({
            'Evaluator': name,
            'Selected Features': ', '.join(selected_features),
            'ROC': roc,
            'Accuracy': accuracy,
            'Kappa': kappa,
            'Precision': precision,
            'Recall': recall,
            'F-score': f_score
        })

    results_df = pd.DataFrame(results)
    results_df = results_df.sort_values(by='ROC', ascending=False).reset_index(

    return results_df
```

In [13]:
```
AW = Features[0]
RW = Features[1]
WW = Features[2]
```

In [14]:
```
# Example of using the function with a validation set
validation_metrics_RW = predict_and_evaluate(RW)
validation_metrics_RW
```

Out[14]:

	Evaluator	Selected Features	ROC	Accuracy	Kappa	Precision	Recall	F-score
0	Extra Trees Classifier	fixed acidity, volatile acidity, citric acid, ...	0.897418	0.816410	0.595906	0.834989	0.885020	0.859277
1	Random Forest	type, fixed acidity, volatile acidity, citric ...	0.886946	0.806667	0.575610	0.830000	0.873684	0.851282
2	Gradient Boosting Trees	fixed acidity, volatile acidity, citric acid, ...	0.823972	0.751795	0.453025	0.785986	0.835628	0.810047
3	Logistic Regression	type, fixed acidity, volatile acidity, residua...	0.791724	0.720513	0.385015	0.763761	0.808907	0.785686
4	LDA	type, fixed acidity, volatile acidity, residua...	0.791577	0.718974	0.384408	0.765661	0.801619	0.783228
5	Decision Trees	fixed acidity, volatile acidity, chlorides, fr...	0.728193	0.744103	0.453051	0.804132	0.787854	0.795910
6	Ridge Classifier	type, fixed acidity, volatile acidity, residua...	NaN	0.716410	0.376738	0.761503	0.804049	0.782198
7	SVC with linear kernel	volatile acidity, residual sugar, free sulfur ...	NaN	0.721026	0.388901	0.767208	0.803239	0.784810
8	Perceptron	fixed acidity, volatile acidity, citric acid, ...	NaN	0.608205	0.173935	0.702494	0.661538	0.681401
9	Linear SVM	type, fixed acidity, volatile acidity, residua...	NaN	0.718974	0.383300	0.764434	0.804049	0.783741

In [15]:
```
# Example of using the function with a validation set
validation_metrics_WW = predict_and_evaluate(WW)
validation_metrics_WW
```

Out[15]:

	Evaluator	Selected Features	ROC	Accuracy	Kappa	Precision	Recall	F-score
0	Extra Trees Classifier	fixed acidity, volatile acidity, citric acid, ...	0.914230	0.837415	0.618954	0.848256	0.920245	0.882786
1	Random Forest	fixed acidity, volatile acidity, citric acid, ...	0.906495	0.840136	0.627686	0.854147	0.916155	0.884065
2	Gradient Boosting Trees	fixed acidity, volatile acidity, citric acid, ...	0.856480	0.801361	0.534208	0.822976	0.893661	0.856863
3	LDA	fixed acidity, volatile acidity, residual suga...	0.822693	0.767347	0.444489	0.790146	0.885481	0.835101
4	Logistic Regression	fixed acidity, volatile acidity, residual suga...	0.822053	0.768027	0.445219	0.789809	0.887526	0.835821
5	Decision Trees	fixed acidity, volatile acidity, citric acid, ...	0.748984	0.776871	0.498471	0.831633	0.833333	0.832482
6	Ridge Classifier	fixed acidity, volatile acidity, residual suga...	NaN	0.768027	0.440403	0.785650	0.895706	0.837076
7	SVC with linear kernel	fixed acidity, volatile acidity, residual suga...	NaN	0.768707	0.447143	0.790528	0.887526	0.836224
8	Perceptron	fixed acidity, residual sugar, chlorides, free...	NaN	0.641497	0.076014	0.682887	0.860941	0.761646
9	Linear SVM	fixed acidity, volatile acidity, residual suga...	NaN	0.770068	0.447425	0.788809	0.893661	0.837967

In [17]:
```
# Example of using the function with a validation set
validation_metrics_AW = predict_and_evaluate(AW)
validation_metrics_AW
```

Out[17]:

	Evaluator	Selected Features	ROC	Accuracy	Kappa	Precision	Recall	F-score
0	Extra Trees Classifier	fixed acidity, volatile acidity, citric acid, ...	0.897418	0.816410	0.595906	0.834989	0.885020	0.859277
1	Random Forest	type, fixed acidity, volatile acidity, citric ...	0.886946	0.806667	0.575610	0.830000	0.873684	0.851282
2	Gradient Boosting Trees	fixed acidity, volatile acidity, citric acid, ...	0.823972	0.751795	0.453025	0.785986	0.835628	0.810047
3	Logistic Regression	type, fixed acidity, volatile acidity, residua...	0.791724	0.720513	0.385015	0.763761	0.808907	0.785686
4	LDA	type, fixed acidity, volatile acidity, residua...	0.791577	0.718974	0.384408	0.765661	0.801619	0.783228
5	Decision Trees	fixed acidity, volatile acidity, chlorides, fr...	0.728193	0.744103	0.453051	0.804132	0.787854	0.795910
6	Ridge Classifier	type, fixed acidity, volatile acidity, residua...	NaN	0.716410	0.376738	0.761503	0.804049	0.782198
7	SVC with linear kernel	volatile acidity, residual sugar, free sulfur ...	NaN	0.721026	0.388901	0.767208	0.803239	0.784810
8	Perceptron	fixed acidity, volatile acidity, citric acid, ...	NaN	0.608205	0.173935	0.702494	0.661538	0.681401
9	Linear SVM	type, fixed acidity, volatile acidity, residua...	NaN	0.718974	0.383300	0.764434	0.804049	0.783741

In [ ]:

STSCI4740_Final_analysis_Aryan.R

jonahkeller

2023-05-05

```
#### Load Libraries ####
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.4.0     v purrr   1.0.1
## v tibble  3.2.1     v dplyr   1.1.0
## v tidyr   1.3.0     v stringr 1.5.0
## v readr   2.1.3     v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
library(ggplot2)
library(gridExtra)
```

```
##
## Attaching package: 'gridExtra'
##
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library(FactoMineR)
library(caret)
```

```
## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(randomForest)
```

```
## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
##
## The following object is masked from 'package:gridExtra':
##
##     combine
##
## The following object is masked from 'package:dplyr':
##
##     combine
##
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(forecast)
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
library(e1071)
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
##
## Attaching package: 'pROC'
##
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
library(kappaSize)
library(stats)
library(ggplot2)
library(corrplot)
library(FactoMineR)
library(dplyr)
library(tidyverse)
library(caret)
library(stringr)
library(caret)
library(e1071)
library(randomForest)
library(gbm)
```

```
## Loaded gbm 2.1.8.1
```

```
library(mlbench)
library(glmnet)
```

```
## Loading required package: Matrix
##
## Attaching package: 'Matrix'
##
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
##
## Loaded glmnet 4.1-7
```

```
library(pROC)
library(tidyr)
library(irr)
```

```
## Loading required package: lpSolve
```

```
library(MLmetrics)
```

```
##
## Attaching package: 'MLmetrics'
##
## The following objects are masked from 'package:caret':
##
##     MAE, RMSE
##
## The following object is masked from 'package:base':
##
##     Recall
```

```
library(kappaSize)
library(MASS)
```

```
##
## Attaching package: 'MASS'
##
## The following object is masked from 'package:dplyr':
##
##     select
```

```
#Importing train/test datasets (see python code for split and explanatory analysis)--------

#Note the data is normalized and we also have a dummy variable for wine quality being good (>=6) or poo

red_wine_train <- read.csv("/Users/jonahkeller/Downloads/Final Project/Data_Aryan/RW_Train.csv") %>% dpl
red_wine_test <- read.csv("/Users/jonahkeller/Downloads/Final Project/Data_Aryan/RW_Test.csv") %>% dplyr
white_wine_train <- read.csv("/Users/jonahkeller/Downloads/Final Project/Data_Aryan/WW_Train.csv") %>% d
white_wine_test <- read.csv("/Users/jonahkeller/Downloads/Final Project/Data_Aryan/WW_Test.csv") %>% dpl
```

```r
all_wine_train <- read.csv("/Users/jonahkeller/Downloads/Final Project/Data_Aryan/AW_Train.csv") %>% dpl
all_wine_test <- read.csv("/Users/jonahkeller/Downloads/Final Project/Data_Aryan/AW_Test.csv") %>% dplyr


#Functions-------------------------------------------------------------------------
format_string <- function(vec){
  vec <- sort(vec)
  vec[2] <- gsub("\\.", " ", vec[2], fixed = TRUE)
  my_str <- paste(vec, collapse = ", ")
  return(my_str)


}
# Model/Variable selection-----------------------------------------------------

column_names <- c("Model","Features Selected","Error/Missclassification Rate","Accuracy")

results_table_red <- matrix(nrow = 0,ncol = 4)
results_table_white <- matrix(nrow = 0,ncol = 4)
colnames(results_table_red) <-column_names
colnames(results_table_white) <-column_names



#Model 1: Logistic Regression
# Variable selection method: Backward selection, Forward Selection (using deviance and AIC)
# Model selection metric: AIC
# stepAIC() function used


#Null and Full models for red wines
null.model_red <- glm(good_quality~1,data = red_wine_train %>% dplyr::select(-type), family = binomial)
full.model_red <- glm(good_quality~.,data = red_wine_train %>% dplyr::select(-type), family = binomial)

#Forward selection for read wines
step.forward_red <- stepAIC(null.model_red,direction = "forward",scope = formula(full.model_red))
```

```
## Start:  AIC=1547.96
## good_quality ~ 1
##
##                        Df Deviance    AIC
## + alcohol               1   1311.1 1315.1
## + volatile.acidity      1   1413.6 1417.6
## + sulphates             1   1416.4 1420.4
## + total.sulfur.dioxide  1   1504.0 1508.0
## + citric.acid           1   1512.9 1516.9
## + density               1   1519.9 1523.9
## + chlorides             1   1523.9 1527.9
## + fixed.acidity         1   1534.7 1538.7
## + free.sulfur.dioxide   1   1542.8 1546.8
## <none>                      1546.0 1548.0
## + pH                    1   1545.6 1549.6
## + residual.sugar        1   1545.9 1549.9
```

```
##
## Step:  AIC=1315.06
## good_quality ~ alcohol
##
##                        Df Deviance    AIC
## + sulphates             1   1216.4 1222.4
## + volatile.acidity      1   1225.8 1231.8
## + fixed.acidity         1   1278.8 1284.8
## + citric.acid           1   1283.7 1289.7
## + pH                    1   1291.9 1297.9
## + total.sulfur.dioxide  1   1302.8 1308.8
## + density               1   1305.7 1311.7
## + residual.sugar        1   1307.6 1313.6
## <none>                      1311.1 1315.1
## + free.sulfur.dioxide   1   1310.8 1316.8
## + chlorides             1   1311.0 1317.0
##
## Step:  AIC=1222.41
## good_quality ~ alcohol + sulphates
##
##                        Df Deviance    AIC
## + volatile.acidity      1   1171.3 1179.3
## + total.sulfur.dioxide  1   1202.0 1210.0
## + fixed.acidity         1   1203.7 1211.7
## + chlorides             1   1209.3 1217.3
## + pH                    1   1209.7 1217.7
## + residual.sugar        1   1212.2 1220.2
## + citric.acid           1   1212.3 1220.3
## <none>                      1216.4 1222.4
## + free.sulfur.dioxide   1   1214.9 1222.9
## + density               1   1216.0 1224.0
##
## Step:  AIC=1179.31
## good_quality ~ alcohol + sulphates + volatile.acidity
##
##                        Df Deviance    AIC
## + total.sulfur.dioxide  1   1158.8 1168.8
## + citric.acid           1   1166.8 1176.8
## + fixed.acidity         1   1167.2 1177.2
## + chlorides             1   1168.3 1178.3
## + residual.sugar        1   1168.6 1178.6
## <none>                      1171.3 1179.3
## + free.sulfur.dioxide   1   1170.2 1180.2
## + pH                    1   1170.7 1180.7
## + density               1   1170.9 1180.9
##
## Step:  AIC=1168.78
## good_quality ~ alcohol + sulphates + volatile.acidity + total.sulfur.dioxide
##
##                        Df Deviance    AIC
## + free.sulfur.dioxide  1   1151.5 1163.5
## + chlorides            1   1155.4 1167.4
## + citric.acid          1   1155.5 1167.5
## + fixed.acidity        1   1156.3 1168.3
```

```
## <none>                     1158.8 1168.8
## + residual.sugar       1   1157.9 1169.9
## + pH                   1   1158.2 1170.2
## + density              1   1158.2 1170.2
##
## Step:  AIC=1163.49
## good_quality ~ alcohol + sulphates + volatile.acidity + total.sulfur.dioxide +
##     free.sulfur.dioxide
##
##                   Df Deviance    AIC
## + fixed.acidity    1   1147.6 1161.6
## + chlorides        1   1148.8 1162.8
## <none>                 1151.5 1163.5
## + pH               1   1149.8 1163.8
## + citric.acid      1   1150.1 1164.1
## + residual.sugar   1   1151.0 1165.0
## + density          1   1151.4 1165.4
##
## Step:  AIC=1161.56
## good_quality ~ alcohol + sulphates + volatile.acidity + total.sulfur.dioxide +
##     free.sulfur.dioxide + fixed.acidity
##
##                   Df Deviance    AIC
## + citric.acid      1   1138.4 1154.4
## + density          1   1141.2 1157.2
## + chlorides        1   1143.2 1159.2
## <none>                 1147.6 1161.6
## + residual.sugar   1   1145.8 1161.8
## + pH               1   1147.5 1163.5
##
## Step:  AIC=1154.37
## good_quality ~ alcohol + sulphates + volatile.acidity + total.sulfur.dioxide +
##     free.sulfur.dioxide + fixed.acidity + citric.acid
##
##                   Df Deviance    AIC
## + density          1   1132.5 1150.5
## + chlorides        1   1135.7 1153.7
## <none>                 1138.4 1154.4
## + residual.sugar   1   1137.3 1155.3
## + pH               1   1138.2 1156.2
##
## Step:  AIC=1150.52
## good_quality ~ alcohol + sulphates + volatile.acidity + total.sulfur.dioxide +
##     free.sulfur.dioxide + fixed.acidity + citric.acid + density
##
##                   Df Deviance    AIC
## <none>                 1132.5 1150.5
## + chlorides        1   1130.6 1150.6
## + pH               1   1131.7 1151.7
## + residual.sugar   1   1132.3 1152.3
```

```
step.forward_red.probs <- predict(step.forward_red,red_wine_test, type = "response")
glm.pred_one <- rep(0,dim(red_wine_test)[1])
glm.pred_one[step.forward_red.probs > .5] <- 1
```

```
table(glm.pred_one,red_wine_test$good_quality)
```

```
##
## glm.pred_one   0   1
##            0 155  71
##            1  68 186
```

```
#Test error rate
er_forward_rw<- mean(glm.pred_one != red_wine_test$good_quality)
features_forward_rw <- names(step.forward_red$coefficients)[-1]


#Backward selection for red wines
step.backward_red <- stepAIC(full.model_red,direction = "backward",scope = formula(null.model_red))
```

```
## Start:  AIC=1153.67
## good_quality ~ fixed.acidity + volatile.acidity + citric.acid +
##     residual.sugar + chlorides + free.sulfur.dioxide + total.sulfur.dioxide +
##     density + pH + sulphates + alcohol
##
##                        Df Deviance    AIC
## - residual.sugar        1   1130.2 1152.2
## - pH                    1   1130.2 1152.2
## - chlorides             1   1131.2 1153.2
## <none>                      1129.7 1153.7
## - free.sulfur.dioxide   1   1134.1 1156.1
## - density               1   1134.8 1156.8
## - citric.acid           1   1136.3 1158.3
## - total.sulfur.dioxide  1   1139.5 1161.5
## - fixed.acidity         1   1142.0 1164.0
## - volatile.acidity      1   1160.3 1182.3
## - alcohol               1   1166.2 1188.2
## - sulphates             1   1193.7 1215.7
##
## Step:  AIC=1152.21
## good_quality ~ fixed.acidity + volatile.acidity + citric.acid +
##     chlorides + free.sulfur.dioxide + total.sulfur.dioxide +
##     density + pH + sulphates + alcohol
##
##                        Df Deviance    AIC
## - pH                    1   1130.6 1150.6
## - chlorides             1   1131.7 1151.7
## <none>                      1130.2 1152.2
## - free.sulfur.dioxide   1   1134.7 1154.7
## - density               1   1135.4 1155.4
## - citric.acid           1   1136.7 1156.7
## - total.sulfur.dioxide  1   1139.6 1159.6
## - fixed.acidity         1   1142.2 1162.2
## - volatile.acidity      1   1160.9 1180.9
## - alcohol               1   1188.9 1208.9
## - sulphates             1   1195.4 1215.4
##
```

```
## Step:  AIC=1150.6
## good_quality ~ fixed.acidity + volatile.acidity + citric.acid +
##     chlorides + free.sulfur.dioxide + total.sulfur.dioxide +
##     density + sulphates + alcohol
##
##                        Df Deviance    AIC
## - chlorides             1   1132.5 1150.5
## <none>                      1130.6 1150.6
## - free.sulfur.dioxide   1   1135.5 1153.5
## - density               1   1135.7 1153.7
## - citric.acid           1   1137.8 1155.8
## - total.sulfur.dioxide  1   1141.0 1159.0
## - fixed.acidity         1   1148.0 1166.0
## - volatile.acidity      1   1161.3 1179.3
## - sulphates             1   1195.5 1213.5
## - alcohol               1   1202.3 1220.3
##
## Step:  AIC=1150.52
## good_quality ~ fixed.acidity + volatile.acidity + citric.acid +
##     free.sulfur.dioxide + total.sulfur.dioxide + density + sulphates +
##     alcohol
##
##                        Df Deviance    AIC
## <none>                      1132.5 1150.5
## - free.sulfur.dioxide   1   1137.5 1153.5
## - density               1   1138.4 1154.4
## - citric.acid           1   1141.2 1157.2
## - total.sulfur.dioxide  1   1142.7 1158.7
## - fixed.acidity         1   1150.0 1166.0
## - volatile.acidity      1   1168.4 1184.4
## - sulphates             1   1195.8 1211.8
## - alcohol               1   1215.8 1231.8
```

```
step.backward_red.probs <- predict(step.backward_red,red_wine_test, type = "response")
glm.pred_two <- rep(0,dim(red_wine_test)[1])
glm.pred_two[step.backward_red.probs > .5] <- 1
table(glm.pred_two,red_wine_test$good_quality)
```

```
##
## glm.pred_two   0   1
##            0 155  71
##            1  68 186
```

```
#Test error rate
er_backward_rw <- mean(glm.pred_two != red_wine_test$good_quality)
features_backward_rw <- names(step.backward_red$coefficients)[-1]

#Null and Full models for red wines
null.model_white <- glm(good_quality~1,data = white_wine_train %>% dplyr::select(-type), family = binomi
full.model_white <- glm(good_quality~.,data = white_wine_train %>% dplyr::select(-type), family = binomi

#Forward selection for white wines
step.forward_white <- stepAIC(null.model_white,direction = "forward",scope = formula(full.model_white))
```

```
## Start:  AIC=4373.3
## good_quality ~ 1
##
##                         Df Deviance    AIC
## + alcohol                1   3860.0 3864.0
## + density                1   4089.3 4093.3
## + chlorides              1   4171.6 4175.6
## + volatile.acidity       1   4183.2 4187.2
## + total.sulfur.dioxide   1   4276.2 4280.2
## + fixed.acidity          1   4345.9 4349.9
## + residual.sugar         1   4349.2 4353.2
## + pH                     1   4352.8 4356.8
## + sulphates              1   4366.0 4370.0
## + free.sulfur.dioxide    1   4368.2 4372.2
## <none>                       4371.3 4373.3
## + citric.acid            1   4370.2 4374.2
##
## Step:  AIC=3864.04
## good_quality ~ alcohol
##
##                         Df Deviance    AIC
## + volatile.acidity       1   3641.7 3647.7
## + free.sulfur.dioxide    1   3782.5 3788.5
## + residual.sugar         1   3814.8 3820.8
## + sulphates              1   3847.8 3853.8
## + chlorides              1   3848.8 3854.8
## + citric.acid            1   3849.5 3855.5
## + fixed.acidity          1   3852.6 3858.6
## + density                1   3855.4 3861.4
## <none>                       3860.0 3864.0
## + pH                     1   3859.5 3865.5
## + total.sulfur.dioxide   1   3859.7 3865.7
##
## Step:  AIC=3647.7
## good_quality ~ alcohol + volatile.acidity
##
##                         Df Deviance    AIC
## + residual.sugar         1   3570.0 3578.0
## + free.sulfur.dioxide    1   3580.2 3588.2
## + sulphates              1   3631.4 3639.4
## + fixed.acidity          1   3631.7 3639.7
## + density                1   3632.4 3640.4
## + total.sulfur.dioxide   1   3633.9 3641.9
## + chlorides              1   3635.2 3643.2
## <none>                       3641.7 3647.7
## + citric.acid            1   3640.9 3648.9
## + pH                     1   3641.7 3649.7
##
## Step:  AIC=3577.98
## good_quality ~ alcohol + volatile.acidity + residual.sugar
##
##                         Df Deviance    AIC
## + density                1   3524.9 3534.9
## + free.sulfur.dioxide    1   3538.9 3548.9
```

```
## + fixed.acidity         1   3556.7 3566.7
## + sulphates             1   3558.7 3568.7
## + chlorides             1   3567.2 3577.2
## <none>                      3570.0 3578.0
## + pH                    1   3569.2 3579.2
## + total.sulfur.dioxide  1   3569.7 3579.7
## + citric.acid           1   3569.9 3579.9
##
## Step:  AIC=3534.85
## good_quality ~ alcohol + volatile.acidity + residual.sugar +
##     density
##
##                         Df Deviance    AIC
## + sulphates              1   3494.4 3506.4
## + free.sulfur.dioxide    1   3498.8 3510.8
## + pH                     1   3517.3 3529.3
## + total.sulfur.dioxide   1   3521.6 3533.6
## + citric.acid            1   3522.8 3534.8
## <none>                       3524.9 3534.9
## + chlorides              1   3524.6 3536.6
## + fixed.acidity          1   3524.8 3536.8
##
## Step:  AIC=3506.36
## good_quality ~ alcohol + volatile.acidity + residual.sugar +
##     density + sulphates
##
##                         Df Deviance    AIC
## + free.sulfur.dioxide    1   3473.1 3487.1
## + pH                     1   3490.2 3504.2
## <none>                       3494.4 3506.4
## + citric.acid            1   3492.4 3506.4
## + fixed.acidity          1   3493.7 3507.7
## + total.sulfur.dioxide   1   3493.7 3507.7
## + chlorides              1   3494.0 3508.0
##
## Step:  AIC=3487.06
## good_quality ~ alcohol + volatile.acidity + residual.sugar +
##     density + sulphates + free.sulfur.dioxide
##
##                         Df Deviance    AIC
## + total.sulfur.dioxide   1   3466.4 3482.4
## + pH                     1   3470.6 3486.6
## <none>                       3473.1 3487.1
## + citric.acid            1   3471.8 3487.8
## + fixed.acidity          1   3472.0 3488.0
## + chlorides              1   3472.4 3488.4
##
## Step:  AIC=3482.43
## good_quality ~ alcohol + volatile.acidity + residual.sugar +
##     density + sulphates + free.sulfur.dioxide + total.sulfur.dioxide
##
##                 Df Deviance    AIC
## + pH             1   3463.9 3481.9
## <none>               3466.4 3482.4
```

```
## + citric.acid    1   3465.1 3483.1
## + fixed.acidity  1   3465.4 3483.4
## + chlorides      1   3466.0 3484.0
##
## Step:  AIC=3481.95
## good_quality ~ alcohol + volatile.acidity + residual.sugar +
##     density + sulphates + free.sulfur.dioxide + total.sulfur.dioxide +
##     pH
##
##                 Df Deviance    AIC
## + fixed.acidity  1   3457.2 3477.2
## + citric.acid    1   3461.4 3481.4
## <none>               3463.9 3481.9
## + chlorides      1   3463.6 3483.6
##
## Step:  AIC=3477.18
## good_quality ~ alcohol + volatile.acidity + residual.sugar +
##     density + sulphates + free.sulfur.dioxide + total.sulfur.dioxide +
##     pH + fixed.acidity
##
##               Df Deviance    AIC
## <none>             3457.2 3477.2
## + citric.acid  1   3455.7 3477.7
## + chlorides    1   3457.1 3479.1
```

```
step.forward_white.probs <- predict(step.forward_white,white_wine_test, type = "response")
glm.pred_three <- rep(0,dim(white_wine_test)[1])
glm.pred_three[step.forward_white.probs > .5] <- 1
table(glm.pred_three,white_wine_test$good_quality)
```

```
##
## glm.pred_three   0   1
##              0 261 112
##              1 231 866
```

```
#Test error rate
er_forward_ww<- mean(glm.pred_three != white_wine_test$good_quality)
features_forward_ww <- names(step.forward_white$coefficients)[-1]

#Backward selection for white wines
step.backward_white <- stepAIC(full.model_white,direction = "backward",scope = formula(null.model_white)
```

```
## Start:  AIC=3479.57
## good_quality ~ fixed.acidity + volatile.acidity + citric.acid +
##     residual.sugar + chlorides + free.sulfur.dioxide + total.sulfur.dioxide +
##     density + pH + sulphates + alcohol
##
##                        Df Deviance    AIC
## - chlorides             1   3455.7 3477.7
## - citric.acid           1   3457.1 3479.1
## <none>                      3455.6 3479.6
## - fixed.acidity         1   3461.0 3483.0
## - total.sulfur.dioxide  1   3462.2 3484.2
```

```
## - pH                   1   3464.1 3486.1
## - alcohol              1   3471.9 3493.9
## - free.sulfur.dioxide  1   3480.1 3502.1
## - sulphates            1   3485.1 3507.1
## - density              1   3499.5 3521.5
## - residual.sugar       1   3534.3 3556.3
## - volatile.acidity     1   3644.9 3666.9
##
## Step:  AIC=3477.66
## good_quality ~ fixed.acidity + volatile.acidity + citric.acid +
##     residual.sugar + free.sulfur.dioxide + total.sulfur.dioxide +
##     density + pH + sulphates + alcohol
##
##                        Df Deviance    AIC
## - citric.acid           1   3457.2 3477.2
## <none>                      3455.7 3477.7
## - fixed.acidity         1   3461.4 3481.4
## - total.sulfur.dioxide  1   3462.4 3482.4
## - pH                    1   3464.5 3484.5
## - alcohol               1   3472.1 3492.1
## - free.sulfur.dioxide   1   3480.2 3500.2
## - sulphates             1   3485.2 3505.2
## - density               1   3502.1 3522.1
## - residual.sugar        1   3540.1 3560.1
## - volatile.acidity      1   3646.4 3666.4
##
## Step:  AIC=3477.18
## good_quality ~ fixed.acidity + volatile.acidity + residual.sugar +
##     free.sulfur.dioxide + total.sulfur.dioxide + density + pH +
##     sulphates + alcohol
##
##                        Df Deviance    AIC
## <none>                      3457.2 3477.2
## - total.sulfur.dioxide  1   3463.7 3481.7
## - fixed.acidity         1   3463.9 3481.9
## - pH                    1   3465.4 3483.4
## - alcohol               1   3473.9 3491.9
## - free.sulfur.dioxide   1   3482.4 3500.4
## - sulphates             1   3487.4 3505.4
## - density               1   3503.0 3521.0
## - residual.sugar        1   3540.8 3558.8
## - volatile.acidity      1   3657.4 3675.4
```

```
step.backward_white.probs <- predict(step.backward_white,white_wine_test, type = "response")
glm.pred_four <- rep(0,dim(white_wine_test)[1])
glm.pred_four[step.backward_white.probs > .5] <- 1
table(glm.pred_four,white_wine_test$good_quality)
```

```
##
## glm.pred_four   0   1
##             0 261 112
##             1 231 866
```

```r
#Test error rate
er_backward_ww<- mean(glm.pred_four != white_wine_test$good_quality)
features_backward_ww <- names(step.backward_white$coefficients)[-1]

#Forward selection performs better as compared to backward selection. Forward selection for white wines

#Model 2: Lasso
# Lasso performs variable selection, we also have to select tuning parameter through cross validation a

#Implementing Lasso for red wines

# Find the best lambda using cross-validation
x <- red_wine_train %>% dplyr::select(-type)
x <- model.matrix(good_quality~.,x)[,-1]
x <-scale(x)
y <- red_wine_train$good_quality

set.seed(10)
fit<- cv.glmnet(x,y,alpha = 1, family = "binomial")
model_rw <- glmnet(x,y, alpha = 1, family = "binomial",lambda = fit$lambda.min)

coef.glmnet(model_rw)
```

```
## 12 x 1 sparse Matrix of class "dgCMatrix"
##                              s0
## (Intercept)           0.23540236
## fixed.acidity         0.60858515
## volatile.acidity     -0.56322743
## citric.acid          -0.30546945
## residual.sugar        0.05711906
## chlorides            -0.10794158
## free.sulfur.dioxide   0.24966823
## total.sulfur.dioxide -0.39147202
## density              -0.37668493
## pH                    0.07611940
## sulphates             0.67644226
## alcohol               0.73489029
```

```r
x.test <- red_wine_test %>% dplyr::select(-type)
x.test <- model.matrix(good_quality~.,x.test)[,-1]
x.test <- scale(x.test)
probabilities_one <- predict.glmnet(model_rw, newx = x.test)
predicted.classes_one <- ifelse(probabilities_one > 0.5, 1, 0)
# Model accuracy
observed.classes <- red_wine_test$good_quality
er_lasso_rw<- mean(predicted.classes_one != observed.classes)



#Implementing Lasso for white wines
x <- white_wine_train %>% dplyr::select(-type)
x <- model.matrix(good_quality~.,x)[,-1]
x <-scale(x)
```

```r
y <- white_wine_train$good_quality

set.seed(10)
fit<- cv.glmnet(x,y,alpha = 1, family = "binomial")
model_ww <- glmnet(x,y, alpha = 1, family = "binomial",
                lambda = fit$lambda.min)

coef.glmnet(model_ww)
```

```
## 12 x 1 sparse Matrix of class "dgCMatrix"
##                               s0
## (Intercept)           0.92251950
## fixed.acidity         0.13166261
## volatile.acidity     -0.62935481
## citric.acid           0.05152348
## residual.sugar        1.00012263
## chlorides            -0.01903213
## free.sulfur.dioxide   0.28413759
## total.sulfur.dioxide -0.15863663
## density              -1.22868720
## pH                    0.15447077
## sulphates             0.24954776
## alcohol               0.49570341
```

```r
x.test <- white_wine_test %>% dplyr::select(-type)
x.test <- model.matrix(good_quality~.,x.test)[,-1]
x.test <- scale(x.test)
probabilities_two <- predict.glmnet(model_ww, newx = x.test)
predicted.classes_two <- ifelse(probabilities_two > 0.5, 1, 0)
# Model accuracy
observed.classes <- white_wine_test$good_quality
er_lasso_ww<- mean(predicted.classes_two != observed.classes)


#Results table
results_table_red <- results_table_red %>%
  rbind(list("Logistic Regression,Forward Stepwise Selection",format_string(features_forward_rw),(er_fo
  rbind(list("Logistic Regression,Backward Stepwise Selection",format_string(features_backward_rw),(er_
  rbind(list("Lasso","All features included",(er_lasso_rw),(1-er_lasso_rw)))

results_table_white <- results_table_white %>%
  rbind(list("Logistic Regression,Forward Stepwise Selection",format_string(features_forward_ww),(er_fo
  rbind(list("Logistic Regression,Backward Stepwise Selection",format_string(features_backward_ww),(er_
  rbind(list("Lasso","All features included",(er_lasso_ww),(1-er_lasso_ww)))
```