STSCI-4740 Final Project Report
Author: Mohamed Abdelrahman (mfa39)

Introduction

The goal of this project was to use the physiochemical properties, found in the Wine Quality Dataset from the UCI Machine Learning Repository, as predictors to predict the quality of red and white variants of the Portuguese "Vinho Verde" wine using various machine learning methods. The machine learning methods used for this analysis were multinomial logistic regression, linear discriminant analysis (LDA), quadratic discriminant analysis (QDA), Naïve Bayes, K-Nearest Neighbors (KNN), and support vector machines (SVMs). All machine learning methods were applied twice, with the first application utilizing a subset of the physiochemical variables, chosen based off correlative properties (see **Data Preprocessing**), as predictors; six physiochemical variables were chosen in total. The second application utilized principal components analysis (PCA) on the physiochemical variables, and the first four principal components were chosen as predictors – this was done to reduce the dimensions of the predictor space while also attempting to improve prediction accuracy. An 80/20 validation set approach was used to train the machine learning models and produce test misclassification error rates which would be used as a metric to compare the performance of models to each other. Overall, the best performing machine learning method in predicting the quality of wine was SVMs utilizing a subset of the physiochemical variables as predictors since it produced the lowest test misclassification error rate of 0.3918399 as seen in *Table 1* below.

Table 1 – Test Misclassification Error Rates

	Multinomial Logistic Regression	*LDA*	*QDA*	*Naïve Bayes*	*KNN*	*SVMs*
Subset of Physiochemical Variables	*0.5011547*	*0.5019246*	*0.5273287*	*0.4996151*	*0.4272517*	*0.3918399*
First Four Principal Components from PCA of Physiochemical Variables	*0.5327175*	*0.5342571*	*0.5288684*	*0.5450346*	*0.4603541*	*0.5304080*

Data Preprocessing

The UCI's Wine Quality Dataset contains 12 variables physiochemical variables along with the response variable quality and a total of 6497 observations for each of the 13 variables. To get an understanding of the distributions of the12 physiochemical variables and the response variable quality, histograms of each of the variables were plotted (see **S1 in Supplementary Materials**). Apart from the type variable, which was transformed into a binary variable with 1 indicating white wine (4898 observations) and 0 indicating red wine (1599 observations), most variables' distributions appeared to be either approximately normal in distribution or extremely

right-skewed. The response variable quality's distribution was of concern, however, since it appeared to have severe class imbalance with the vast majority of observations having quality labels of either 5, 6, or 7 as seen below in Table 2.

Table 2 – Distribution of Class Labels for Response Variable quality

3	*4*	*5*	*6*	*7*	*8*	*9*
30	*216*	*2138*	*2836*	*1079*	*193*	*5*

The reason severe class imbalance is problematic in a classification problem is because when training our machine learning models, they will be more biased toward the majority classes since there are more observations to train on and hence will have poor predictive performance, especially when attempting to predict the minority class **[5]**. To fix this severe class imbalance for the sake of enhancing our future predictive performance, all 30 observations with quality label of 3 were re-labeled as having quality label 4, hence, the number of quality label 4 observations became 246. The same was done for the 5 observations with quality label 9 as they were re-labeled as having quality label 8, bringing the number of observations with quality label 8 to 198. While this may not completely fix the class imbalance as there are still majority classes (5, 6, 7) and minority classes (4, 8), it will alleviate the problem greatly while relatively keeping the authenticity of the dataset.

To begin the analysis, the dataset was split into training and testing datasets by utilizing an 80/20 validation set approach, such that a random 80% of the observations will be used to build a model(s) and train the machine learning methods, while the other remaining 20% will be used to test the machine learning models' predictive power by obtaining test misclassification error rates for each model. A seed of 21 was set in order to ensure the reproducibility of the results and this same seed was utilized in all following machine learning analysis performed in this report. In order to choose predictors to build our models, the training dataset was used to generate a correlation plot (see *Figure 1*) and a correlation matrix (see **S2 in Supplementary Materials**) of the 12 physiochemical variables and the response variable quality.

Figure 1 – Correlation Plot of Variables

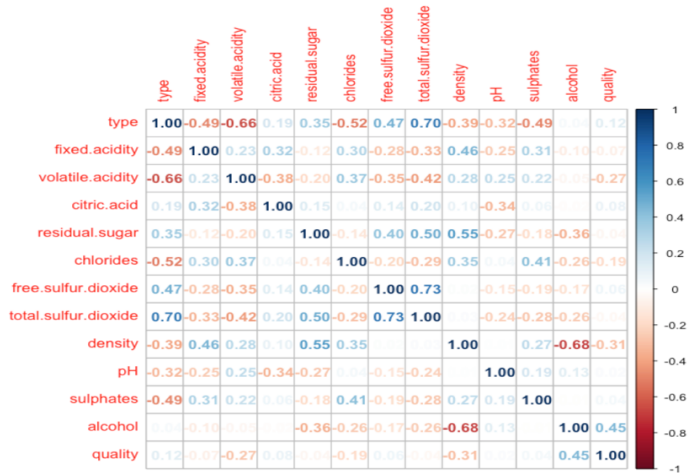


Figure 1 – Correlation Plot of Variables: The figure above depicts the correlation of the physiochemical variables with each and with the response variable quality. It is evident, apart from a few cases, the physiochemical variables are not too highly correlated with each other or with the response variable quality.

Originally, any physiochemical variable with an absolute value of correlation greater than 0.05 with quality was chosen as a predictor, however, to avoid problems caused by multicollinearity, one of any two predictors who have an absolute value of correlation greater than 0.6 would be dropped and the other would be kept **[2]**. In the end a total of 6 out of 12 of the physiochemical variables were kept and used to model quality – these variables were alcohol, volatile.acidity, chlorides, fixed.acidity, citric.acid, and free.sulfur.dioxide.

The training dataset was also utilized to perform PCA on the 12 physiochemical variables (see **S3 in Supplementary Materials**) with the goal being to model quality on a subset of the 12 principal components that would result in a model with less than 6 predictors, due to the fear that some of our machine learning methods would break down with high numbers of predictors or have their predictive accuracy decrease due to the method suffering from the curse of dimensionality **[2, 6]**. We chose our final model with the help of a Scree plot (see *Figure 2*) and modeled quality on the first four principal components which explained approximately 74% of the total variation found among the 12 physiochemical variables **[4]**. It was assumed this model would outperform the model based on a subset of the physiochemical variables since it is lower in dimension and has uncorrelated predictors, hence, it would not cause some methods to break down / suffer from the curse of dimensionality and / or from multicollinearity **[2, 6]**.

Figure 2 – Scree Plot for Principal Components Selection

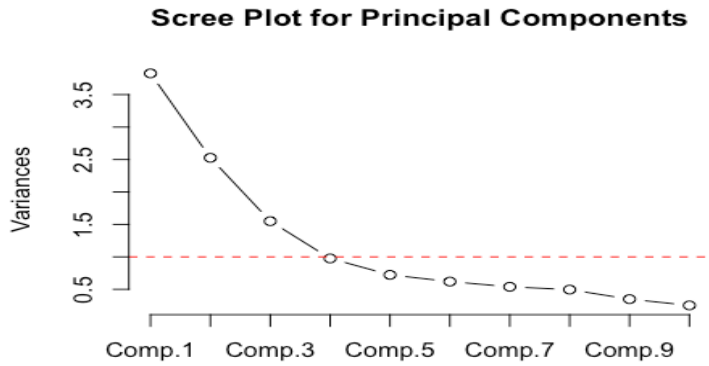


Figure 2 – Scree Plot for Principal Components Selection: The figure above is a Scree Plot that plots the principal components on the x-axis and their respective variances (eigenvalues) on the y-axis. It is advised to include principal components whose variance is approximately greater than or equal to 1 (red line). The first four principal components were chosen utilizing this plot.

After using the training dataset to choose predictors based off correlation between the 12 physiochemical variables and principal components, two models (1) and (2) were chosen. Model (1) is based off modeling quality on a subset of the 12 physiochemical variables, and model (2) models' quality off the first four principal components.

(1) quality ~ alcohol + volatile.acidity + chlorides + fixed.acidity + citric.acid + free.sulfur.dioxide
(2) quality ~ PC1 + PC2 + PC3 + PC4

Multinomial Logistic Regression

The first machine learning method applied was multinomial logistic regression, which can be used when the response variable has more than two classes. The function multinom found in the package nnet was used to perform this analysis. The training dataset along with the multinom function was used to train two models, the first model (1) using the 6 physiochemical variables as predictors and the second model (2) using the first 4 principal components as predictors. The test dataset along with the predict function was used to make predictions and calculate the test misclassification error rate for both model 1 and 2. Model 1 achieved a test misclassification error rate of 0.5011547 and model 2 achieved a test misclassification error rate of 0.5327175. Overall, both multinomial logistic regression models performed extremely poor, often misclassifying observations than correctly classifying them.

Linear Discriminant Analysis

LDA is another extremely popular machine learning method often used over logistic regression when the response variable has more than two classes, the classes have identical variance-covariance matrices, there is a good amount of separation between the classes, and the predictors are drawn from Gaussian distributions. Although the third assumption cannot be completely validated for the predictors in the Wine Quality Dataset, since one cannot plot in 6- or 4-dimensional space, nor is the fourth assumption completely valid as seen in the histogram plots (see **S1 in Supplementary Materials**), all assumptions were assumed to be true and LDA was applied using the function lda found in the package MASS. The training dataset along with the lda function was used to train two models, the first model (1) using the 6 physiochemical variables as predictors and the second model (2) using the first 4 principal components as predictors. The test dataset along with the predict function was used to make predictions and calculate the test misclassification error rate for both model 1 and 2. Model 1 achieved a test misclassification error rate of 0.5019246 and model 2 achieved a test misclassification error rate of 0.5342571. The LDA model failed to outperform the multinomial logistic regression model most likely due to its strict assumptions and, hence, lower flexibility. Overall, both LDA models performed extremely poor, often misclassifying observations than correctly classifying them.

Quadratic Discriminant Analysis

QDA is another extremely popular machine learning method that is almost identical to LDA, since they share many of the same assumptions, except that QDA relaxes one of the main assumptions of LDA which states that all classes have identical variance-covariance matrices, by assuming each the class's variance-covariance matrices are not identical, i.e., each class has its own variance-covariance matrix. All of QDA's assumptions were assumed to be met and QDA was applied using the qda function found in the package MASS. The training dataset along with the qda function was used to train two models, the first model (1) using the 6 physiochemical variables as predictors and the second model (2) using the first 4 principal components as predictors. The test dataset along with the predict function was used to make predictions and calculate the test misclassification error rate for both model 1 and 2. Model 1 achieved a test misclassification error rate of 0.5273287 and model 2 achieved a test misclassification error rate of 0.5288684. Contrary to initial assumptions, the relaxed variance-covariance assumption of

QDA models did not improve upon the LDA models and in fact performed worse. Overall, both QDA models performed extremely poor, often misclassifying observations than correctly classifying them.

Naïve Bayes

In an attempt to improve upon both LDA and QDA misclassification test error rates the Naïve Bayes classifier was an extremely attractive option to use. Naïve Bayes assumes that all predictors are independent of each other within a class, hence, the joint distribution can be derived from the product of the marginals. Furthermore, Naïve Bayes does not assume the predictors are drawn from a Gaussian distribution and will tend to estimate the predictors individual distributions through a kernel density estimator **[2]**. Although it is evident from the correlation between the predictors that the first assumption does not hold well, it was assumed that because Naïve Bayes could be performed with kernel density estimates of each predictor's distribution, we could greatly reduce the test misclassification error rate, since it was evident from the predictors' histograms (see **S1 in Supplementary Materials**) that the vast majority were not Gaussian distributed. All of Naïve Bayes' assumptions were assumed to be met, and Naïve Bayes was applied the first time under the assumptions the predictors were Gaussian distributed using the naiveBayes function in the package e1071, and again a second time using kernel estimated densities using the function NaiveBayes in the package klaR. The training dataset along with both Naïve Bayes functions (naiveBayes and NaiveBayes) was used to train four models - the first model (1) using the 6 physiochemical variables as predictors under the assumption of Gaussian densities, the second model (2) using the first 4 principal components as predictors under the assumption of Gaussian densities, the third model (3) using the 6 physiochemical variables as predictors with kernel estimated densities, and the fourth model (4) using the first 4 principal components as predictors with kernel estimated densities. The test dataset along with the predict function was used to make predictions and calculate the test misclassification error rate for models 1, 2, 3, and 4 (see *Table 3*). Model 1 achieved a test misclassification error rate of 0.5419554, model 2 achieved a test misclassification error rate of 0.5488838, model 3 achieved a test misclassification error rate of 0.4996151, and model 4 achieved a test misclassification error rate of 0.5450346. Apart from model 3 which was a slight improvement from the test misclassification error rates obtained by QDA and LDA, the Naïve Bayes classifier, contrary to assumptions, performed worser than LDA and QDA. This poor performance could be due to the independence assumptions since it was obvious from the correlation plot (see *Figure 1*) the none of the physiochemical are independent. The failed independence assumption is also explanatory of the PCA based models' poor performance, since although the principal components are independent, they are made of linear combinations of correlated variables. Overall, all four Naïve Bayes models performed extremely poor, with three out of the four models often misclassifying observations than correctly classifying them.

Table 3 – Test Misclassification Error Rates for Naïve Bayes

	Gaussian Density	*Kernel Estimated Density*
Subset of Physiochemical Variables	*0.5419554*	*0.4996151*
First Four Principal Components from PCA of Physiochemical Variables	*0.5488838*	*0.5450346*

K-Nearest Neighbors

The vast majority of machine learning methods discussed in this paper have been parametric methods and have yielded extremely poor test misclassification error rates. In an attempt to improve the test misclassification error rate, the non-parametric machine learning algorithm KNN was applied. KNN is a non-parametric, supervised machine learning algorithm that assumes that similar points can be found near each other, and hence utilizes proximity to classify clusters / groups of points in predictor space into a class. Although KNN is a non-parametric method, and based on the value of the hyperparameter k, the number of neighboring points near an observation to consider for classifying the observation, can be more flexible than the previously applied methods it does suffer from one fatal draw back – the curse of dimensionality **[7]**. Generally, when the number of predictors is greater than 4, the predictor space becomes increasingly sparse, hence, leading to the breakdown of KNN's prediction accuracy.

The function knn found in the package class was used to perform the KNN analysis. The training dataset along with the knn function was used to train two models, each on a range of k values ranging from 1-100. The first model (1) used the 6 physiochemical variables as predictors and the second model (2) used the first 4 principal components as predictors. The test dataset along with the predict function was used to make predictions and calculate the test misclassification error rate for both models 1 and 2 for each value of k. For both models, the value of k = 1 achieved the lowest test misclassification error rate. Model 1 achieved a test misclassification error rate of 0.4272517 and model 2 achieved a test misclassification error rate of 0.4603541. It was initially assumed model 2 would outperform model 1 since its predictor space was lower in dimensions (4 vs. 6). However, since both models minimized the test misclassification error rate using a value of k = 1, it is reasonable that model 1 would outperform model 2, even though model 2 has lesser dimensions, since one of the main downsides of PCA is that it results in a loss of information **[2]**.

Support Vector Machines

The final machine learning method applied in this analysis was a highly flexible, non-parametric, supervised learning method known as SVMs. SVMs are a broader generalization of the maximal margin classifier, who seeks to place a hyperplane between points in p-dimensional space such that points on one side of the hyperplane are classified as one class and points on the other are classified into the other class (in the case of binary classes) **[2]**. However, this separating hyperplane is generally not available, hence, SVMs tend to attempt to marginalize soft margins, such that "support vectors" (p-dimensional points) can be located on the hyperplane or even on the wrong side of the hyperplane at the expense of a hyperparameter cost – the maximum number of support vectors on the wrong side of the hyperplane. The cost hyperparameter's value can be chosen by methods such as k-fold cross validation and is a reflection of the bias-variance trade off, with smaller values leading to more flexible models.

SVMs also utilize kernels, functions which are generalizations of inner products that indicate the level of similarity between two observations, to generate linear and non-linear, such as polynomial and radial, decision boundaries. Although SVMs are generally used for classifying binary class labels, they can be extended to classification settings with k-classes by utilizing a

one-versus-one, all pairs approach that compares all possible combinations of pairs of classes to each other and assigns an observation to the class that was classified to the most **[2]**.

Based off the nature of the data, it was decided that we would perform SVM analysis twice, the first with a linear kernel and the second with a radial kernel since we suspected the true decision boundary was either linear or radial. To begin our SVM analysis, the functions svm and tune in the package e1071 were used along with the training dataset to train four models using 10-fold cross validation to choose hyperparameters. The first model (1) used a linear kernel and the 6 physiochemical variables as predictors and was trained using with a grid of potential values {0.001, 0.01, 0.1, 1, 5, 10, 100} for the hyperparameter cost; 10-fold cross validation found the best performing model used cost = 0.1. The second model (2) used a linear kernel and the first 4 principal components as predictors and was trained using with a grid of potential values {0.001, 0.01, 0.1, 1, 5, 10, 100} for the hyperparameter cost; 10-fold cross validation found the best performing model used cost = 0.01. The third model (3) used a radial kernel and the 6 physiochemical variables as predictors and was trained using with a grid of potential values {0.1, 1, 10, 100, 1000} for the hyperparameter cost and another grid of potential values {0.5, 1, 2, 3, 4} for the hyperparameter gamma (controls the non-linearity of the fit); 10-fold cross validation found the best performing model used cost = 10 and gamma = 4. The fourth model (4) used a radial kernel and the first 4 principal components as predictors and was trained using with a grid of potential values {0.1, 1, 10, 100, 1000} for the hyperparameter cost and another grid of potential values {0.5, 1, 2, 3, 4} for the hyperparameter gamma; 10-fold cross validation found the best performing model used cost = 1000 and gamma = 3 (see **S4 in Supplementary Materials**).

The test dataset along with the predict function was used to make predictions and calculate the test misclassification error rate for models 1, 2, 3, and 4 (see *Table 4*). Model 1 achieved a test misclassification error rate of 0.4934565, model 2 achieved a test misclassification error rate of 0.5327175, model 3 achieved a test misclassification error rate of 0.3918399, and model 4 achieved a test misclassification error rate of 0.5304080. Model 3 achieved the lowest test misclassification error rate out of all machine learning methods analyzed in this study, and although model 1 slightly outperformed all other machine learning methods apart from KNN in this report, the difference was not too drastic. Overall, the best performing machine learning method in this report was a radial kernel based SVM with cost = 10 and gamma = 4 using the 6 physiochemical variables as predictors.

Table 4 – Test Misclassification Error Rates for SVMs

	Linear Kernel	*Radial Kernel*
Subset of Physiochemical Variables	*0.4934565*	*0.3918399*
First Four Principal Components from PCA of Physiochemical Variables	*0.5327175*	*0.5304080*

Concluding Remarks

In this report we applied the machine learning methods multinomial logistic regression, linear discriminant analysis (LDA), quadratic discriminant analysis (QDA), Naïve Bayes, K-Nearest Neighbors (KNN), and support vector machines (SVMs) on the UCI Machine Learning Repository's Wine Quality Dataset, with an 80/20 validation set approach to predict the response variable quality. Overall, SVMs produced the best performing machine learning method in this report achieving a test misclassification error rate of 0.3918399. However, their high flexibility and relatively low interpretability may not make them as appealing to use as other methods, since their improvement in accuracy is not drastically consistent across all models trained (see *Table 4*). Nevertheless, this analysis witnessed extremely poor classification results demonstrated by high test misclassification error rates obtained (see *Table 1*), with most test misclassification error rates being around ~ 0.5, i.e., the methods misclassify observations approximately half of the time.

Furthermore, comparing all parametric methods (i.e., multinomial logistic regression, LDA, QDA, and Naïve Bayes) that utilized the six physiochemical variables as predictors (model 1) we tend to observe that Naïve Bayes (with kernel estimated densities) and multinomial logistic regression performed the best each obtaining a test misclassification error rate of 0.4996151 and 0.5011547 respectively (see *Table 1*). This result is quite interesting as the worst performing parametric methods were the relatively more flexible ones such as QDA and LDA obtaining a test misclassification error rate of 0.5273287 and 0.5019246. Although the difference is not huge, this "worser" performance could be due to these relatively more flexible models overfitting on the training dataset and hence performing relatively worse (even if by a slim margin) on the test dataset. This trend, however, is reversed when the first four principal components are used as predictors (model 2) as the relatively most flexible method QDA outperforms all other parametric methods and obtaining a test error misclassification rate of 0.5288684.

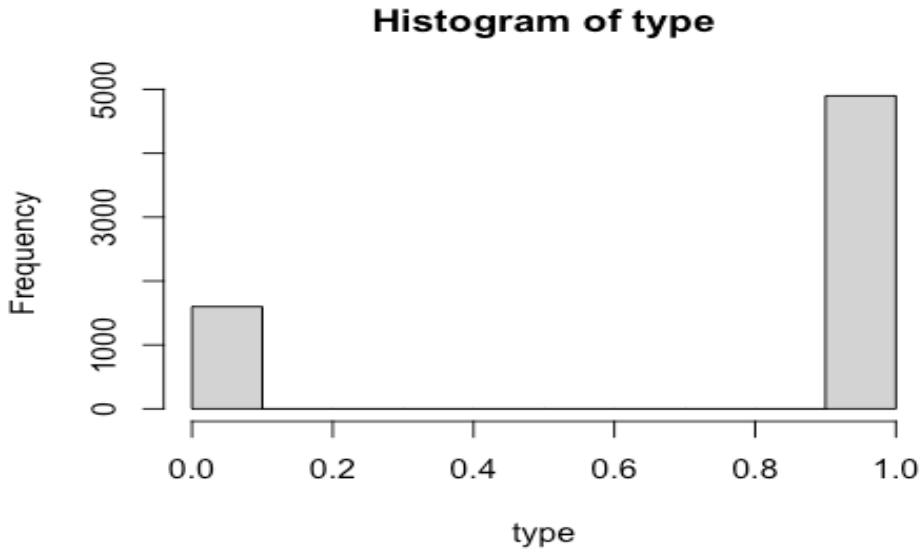
One interesting trend in both parametric and non-parametric methods is the relatively worser performance of the models utilizing the first four principal components as predictors compared to those using the six physiochemical variables as predictors, even though it was initially hypothesized they would perform better due to them having no correlations between principal components (no multicollinearity) and a lower dimensional predictor space (limits breakdown of methods due to curse of dimensionality). On second look, however, due to the low correlation of the physiochemical variables with each other and with the response, it would be no surprise using the first 4 principal components would result in worser performing models since they can only explain ~ 74% of the variation within the physiochemical variables, who themselves, lack correlation with each other. It is more than likely due to the sparse nature of the physiochemical variables and their lack of correlation with each other and the response, multicollinearity was never a problem, and performing PCA, an unsupervised learning algorithm, only resulted in the loss of information. Therefore, when utilizing the first four principal components as predictors, the loss of information hindered the prediction accuracy of the machine learning methods - resulting in higher test misclassification error rates for every machine learning method relative to the model with the six physiochemical predictors.

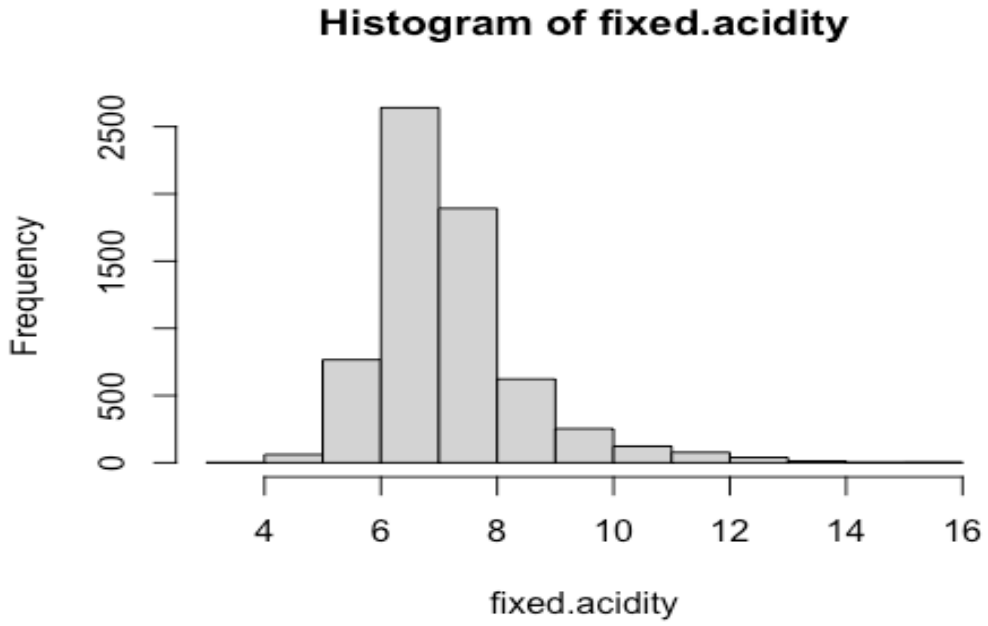
Works Cited

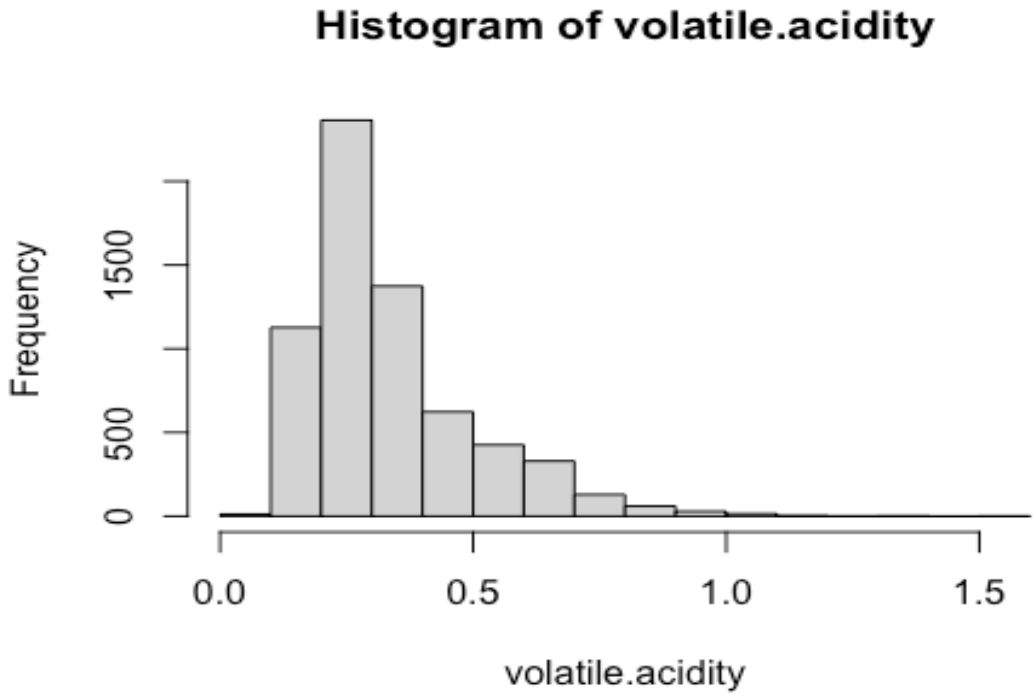
1. Wine Quality Dataset - https://archive.ics.uci.edu/dataset/186/wine+quality
2. ISL - https://www.statlearning.com
3. Naïve Bayes - https://rdrr.io/cran/klaR/man/NaiveBayes.html
4. Scree Plot - https://en.wikipedia.org/wiki/Scree_plot
5. Class Imbalance - https://machinelearningmastery.com/what-is-imbalanced-classification/ & https://developers.google.com/machine-learning/data-prep/construct/sampling-splitting/imbalanced-data
6. Curse of Dimensionality - https://en.wikipedia.org/wiki/Curse_of_dimensionality
7. KNN - https://www.ibm.com/topics/knn#:~:text=Next%20steps-,K%2DNearest%20Neighbors%20Algorithm,of%20an%20individual%20data%20point.

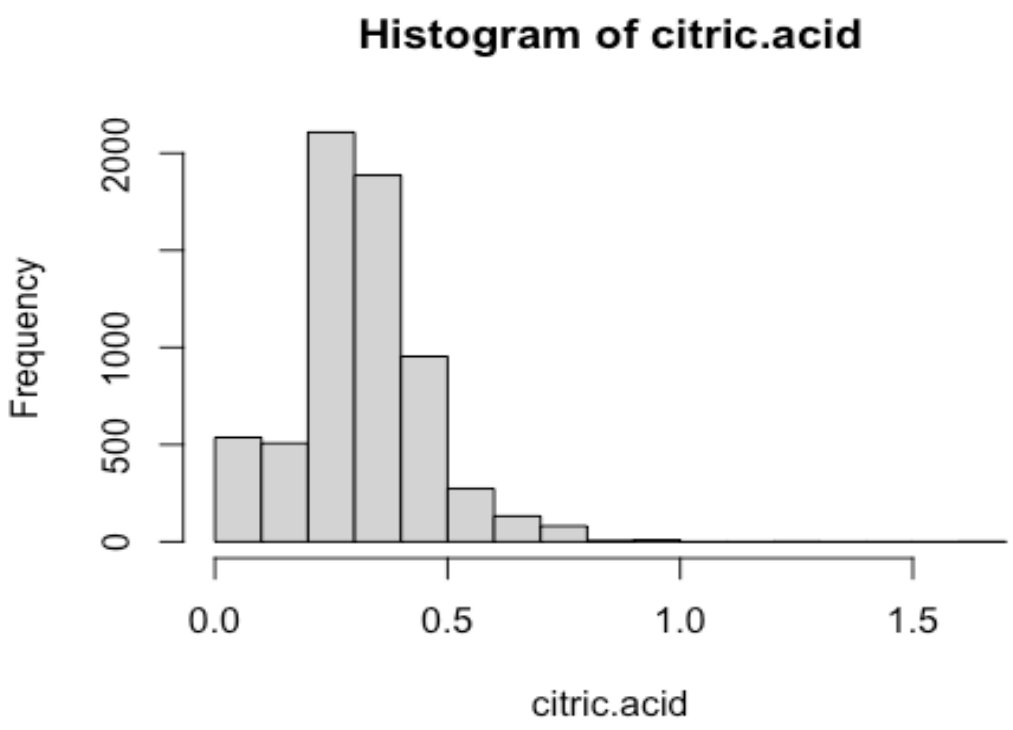
Supplementary Materials

S1 – Histograms of Variables

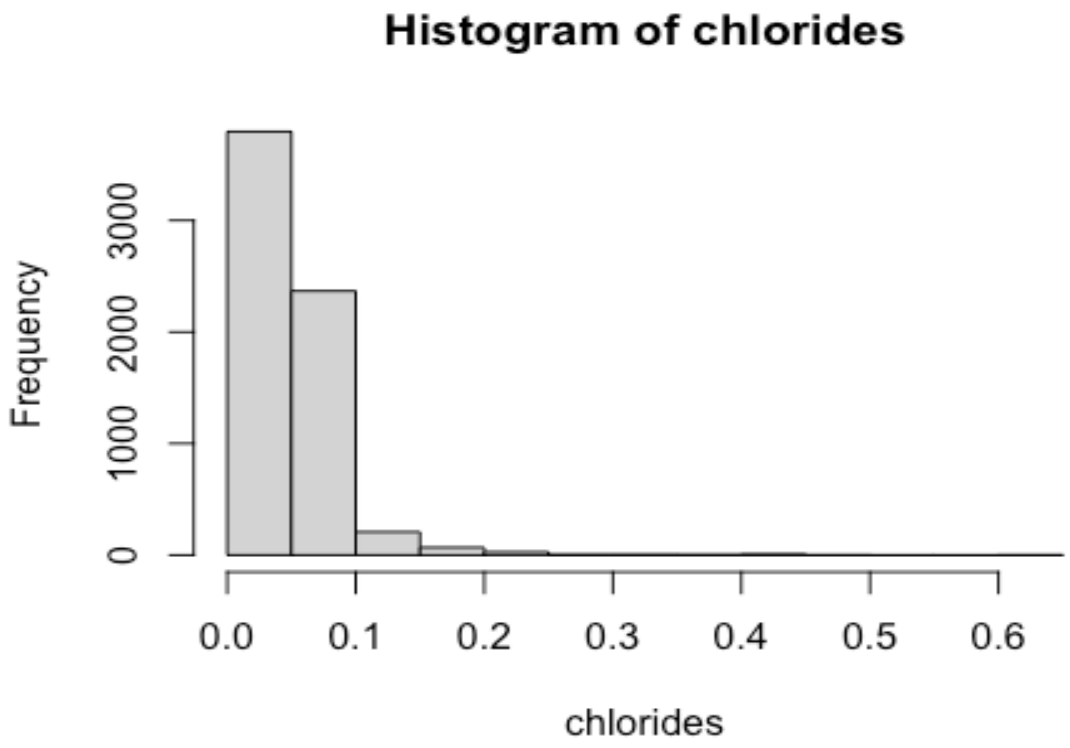


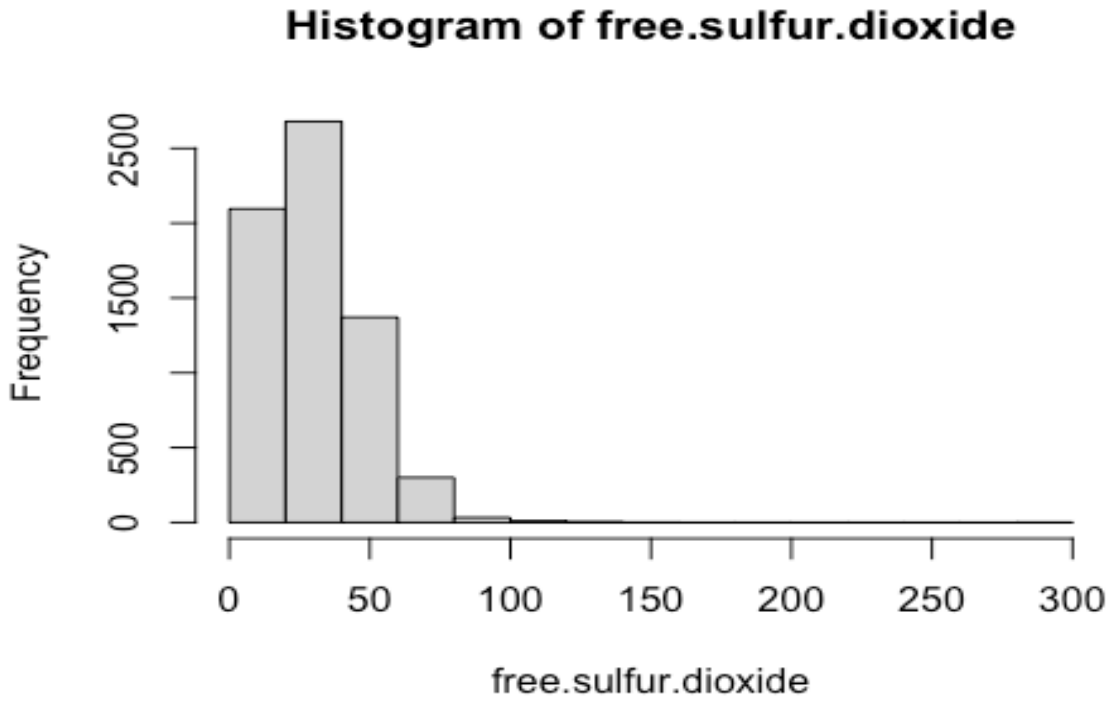


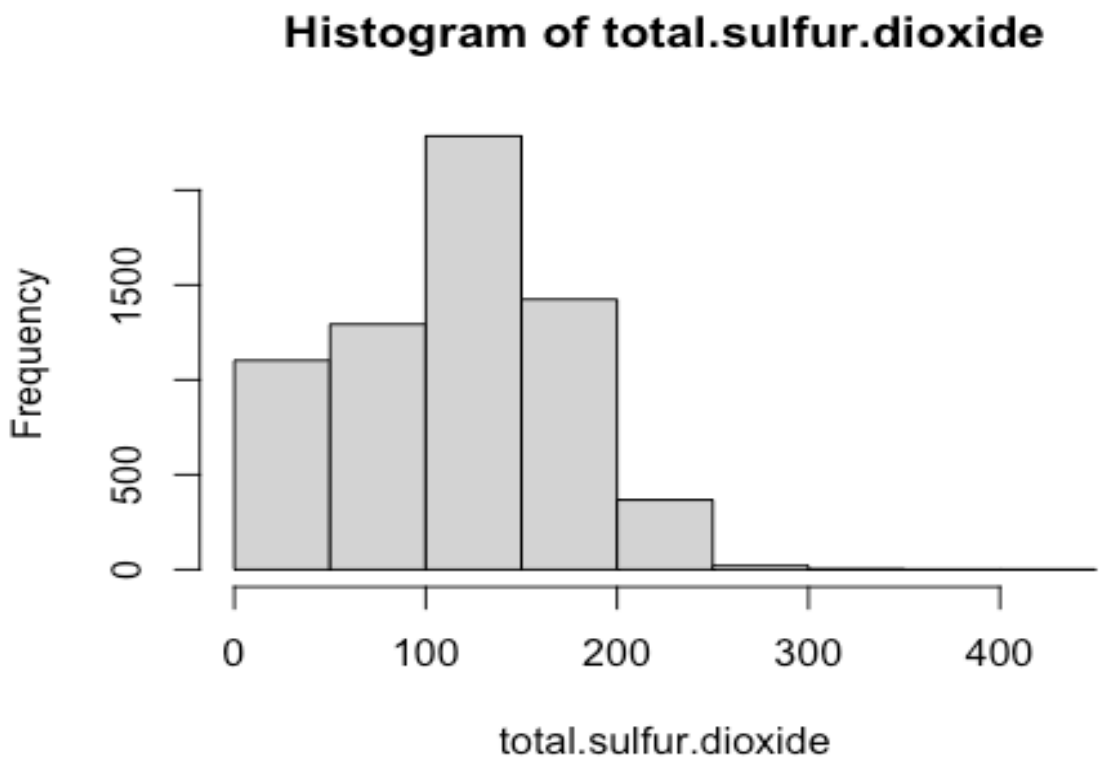

Histogram of volatile.acidity
Frequency
0
500
1500
0.0
0.5
1.0
1.5
volatile.acidity

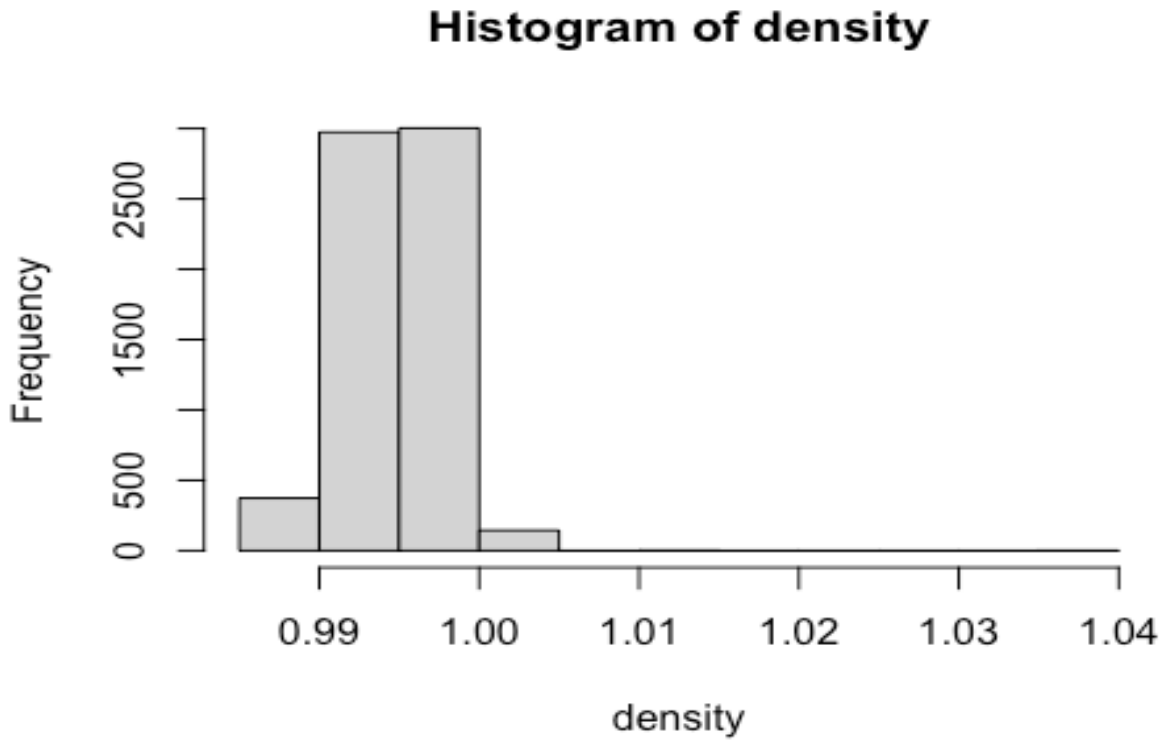

Histogram of citric.acid
Frequency
0
500
1000
2000
0.0
0.5
1.0
1.5
citric.acid

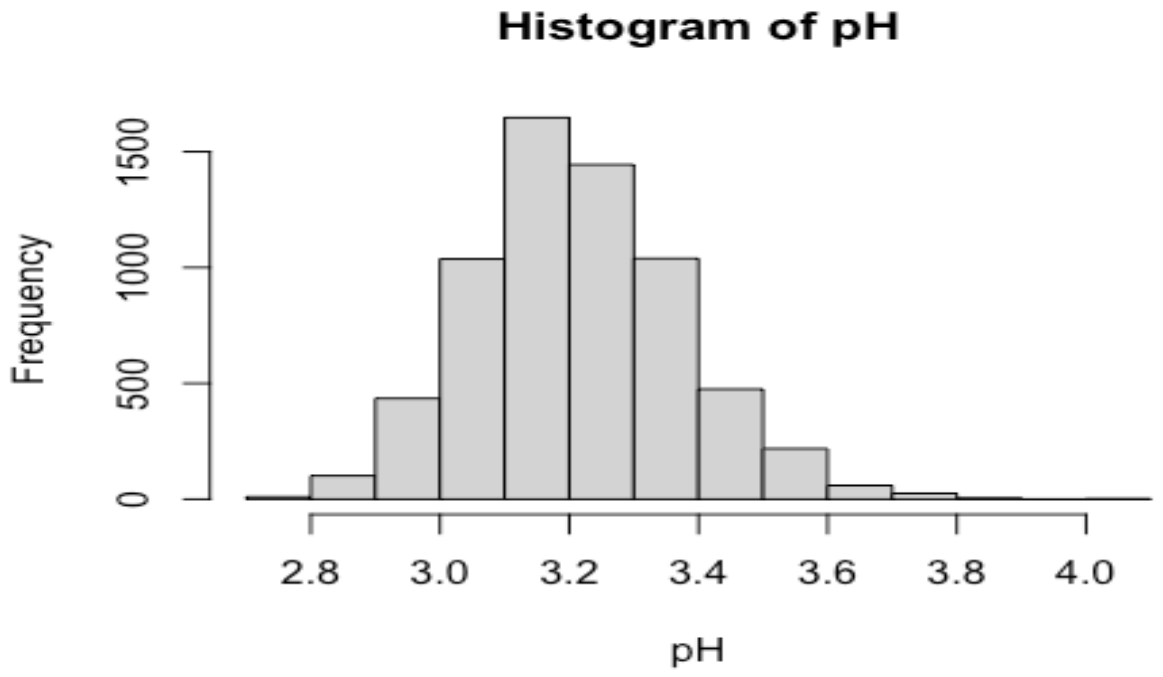

Histogram of residual.sugar
Frequency
0
1000
3000
0
10
20
30
40
50
60
70
residual.sugar

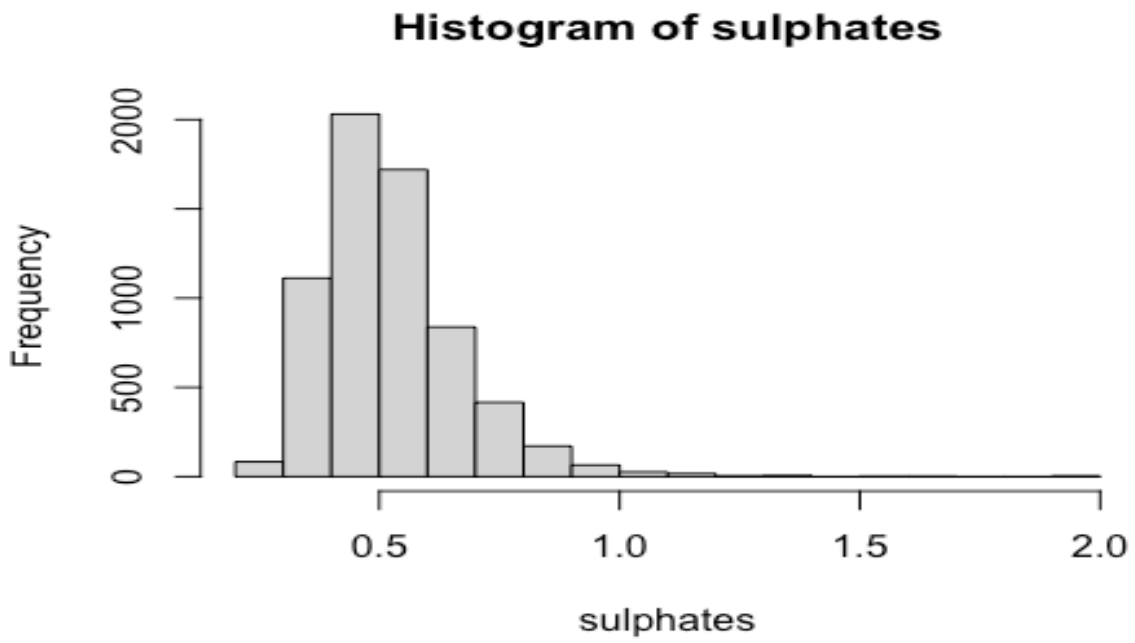

Histogram of chlorides
Frequency
0
1000
2000
3000
0.0
0.1
0.2
0.3
0.4
0.5
0.6
chlorides

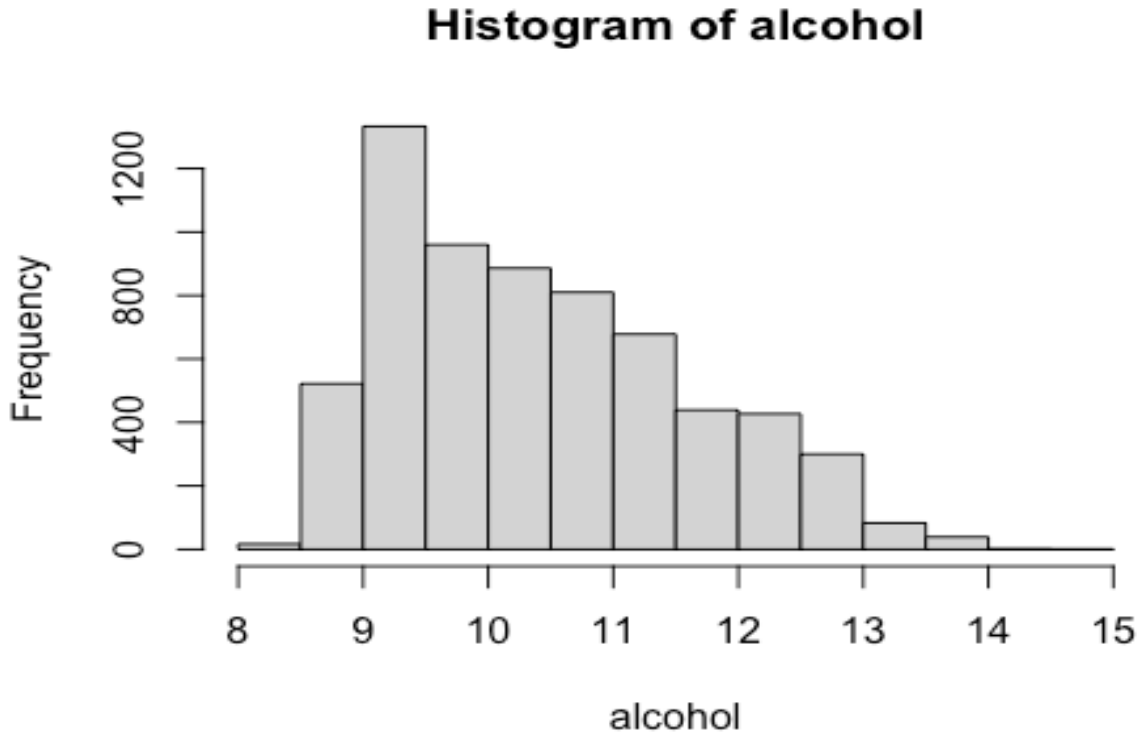

Histogram of free.sulfur.dioxide
Frequency
0
500
1500
2500
0
50
100
150
200
250
300
free.sulfur.dioxide

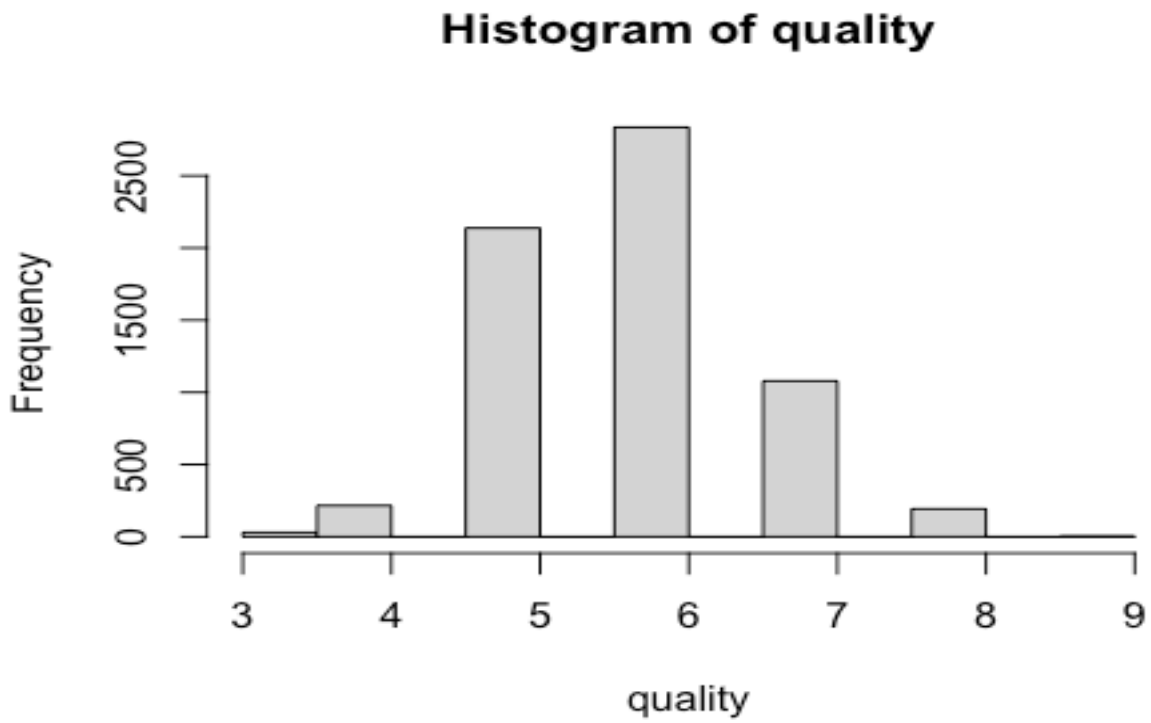

Histogram of total.sulfur.dioxide
Frequency
0
500
1500
0
100
200
300
400
total.sulfur.dioxide


Histogram of density
Frequency
0
500
1500
2500
0.99
1.00
1.01
1.02
1.03
1.04
density


Histogram of pH
Frequency
0
500
1000
1500
2.8
3.0
3.2
3.4
3.6
3.8
4.0
pH


Histogram of sulphates
Frequency
0
500
1000
2000
0.5
1.0
1.5
2.0
sulphates


Histogram of alcohol
Frequency
0
400
800
1200
8
9
10
11
12
13
14
15
alcohol



S2 - Correlation Matrix Between Variables in Training Dataset

	type	fixed.acidity	volatile.acidity	citric.acid	residual.sugar	chlorides	free.sulfur.dioxide	total.sulfur.dioxide
type	1.00000000	-0.48985788	-0.65734718	0.18874429	0.3528340	-0.51597170	0.47076318	0.70343647
fixed.acidity	-0.48985788	1.00000000	0.22519380	0.32114389	-0.1150370	0.29752339	-0.27625743	-0.33338922
volatile.acidity	-0.65734718	0.22519380	1.00000000	-0.37515037	-0.1988695	0.37024972	-0.35224435	-0.42168370
citric.acid	0.18874429	0.32114389	-0.37515037	1.00000000	0.1468076	0.04487782	0.13866517	0.20024791
residual.sugar	0.35283396	-0.11503701	-0.19886955	0.14680760	1.0000000	-0.13857389	0.39535633	0.49597514
chlorides	-0.51597170	0.29752339	0.37024972	0.04487782	-0.1385739	1.00000000	-0.20308285	-0.29129213
free.sulfur.dioxide	0.47076318	-0.27625743	-0.35224435	0.13866517	0.3953563	-0.20308285	1.00000000	0.72539500
total.sulfur.dioxide	0.70343647	-0.33338922	-0.42168370	0.20024791	0.4959751	-0.29129213	0.72539500	1.00000000
density	-0.39130993	0.45952188	0.27503484	0.09935365	0.5517208	0.35457023	0.01738796	0.02561293
pH	-0.32416288	-0.25263527	0.25409292	-0.33519365	-0.2708072	0.04048648	-0.14855934	-0.23727623
sulphates	-0.48920011	0.31010426	0.22329332	0.06429158	-0.1833223	0.41256205	-0.18919694	-0.27875549
alcohol	0.03790065	-0.09905516	-0.04509854	-0.01721328	-0.3583146	-0.25590068	-0.16775467	-0.25810151
quality	0.12360743	-0.07257874	-0.27205317	0.08409299	-0.0429262	-0.19135030	0.06462266	-0.03562948

	density	pH	sulphates	alcohol	quality
type	-0.391309928	-0.324162878	-0.48920011	0.03790065	0.12360743
fixed.acidity	0.459521875	-0.252635269	0.31010426	-0.09905516	-0.07257874
volatile.acidity	0.275034845	0.254092920	0.22329332	-0.04509854	-0.27205317
citric.acid	0.099353653	-0.335193654	0.06429158	-0.01721328	0.08409299
residual.sugar	0.551720817	-0.270807243	-0.18332231	-0.35831457	-0.04292620
chlorides	0.354570229	0.040486479	0.41256205	-0.25590068	-0.19135030
free.sulfur.dioxide	0.017387958	-0.148559338	-0.18919694	-0.16775467	0.06462266
total.sulfur.dioxide	0.025612930	-0.237276229	-0.27875549	-0.25810151	-0.03562948
density	1.000000000	0.006596032	0.26535256	-0.68466083	-0.30891142
pH	0.006596032	1.000000000	0.18546996	0.12512436	0.02183046
sulphates	0.265352555	0.185469964	1.00000000	-0.00626715	0.04394295
alcohol	-0.684660827	0.125124356	-0.00626715	1.00000000	0.44886864
quality	-0.308911416	0.021830456	0.04394295	0.44886864	1.00000000

S3 – PCA on Training Dataset Results

```
## Importance of components:
##                           Comp.1    Comp.2    Comp.3     Comp.4     Comp.5
## Standard deviation     1.9562464 1.5894372 1.2449498 0.98782704 0.85038421
## Proportion of Variance 0.3189083 0.2105259 0.1291583 0.08131686 0.06026278
## Cumulative Proportion  0.3189083 0.5294342 0.6585925 0.73990939 0.80017216
##                            Comp.6     Comp.7     Comp.8     Comp.9   Comp.10
## Standard deviation     0.78698676 0.73417950 0.70397298 0.59072624 0.5025014
## Proportion of Variance 0.05161235 0.04491829 0.04129816 0.02907979 0.0210423
## Cumulative Proportion  0.85178451 0.89670280 0.93800097 0.96708076 0.9881231
##                            Comp.11     Comp.12
## Standard deviation     0.342843282 0.158056189
## Proportion of Variance 0.009795126 0.002081813
## Cumulative Proportion  0.997918187 1.000000000
```

S4 – R Code

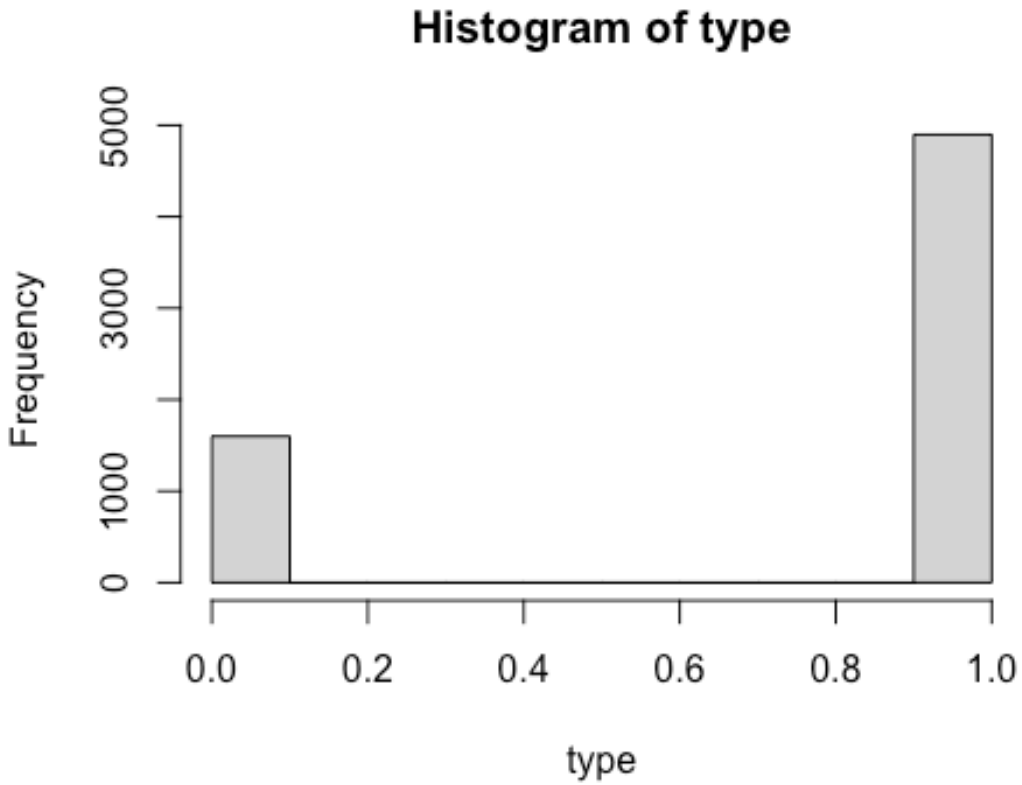
STSCI-4740 Final Project R Code

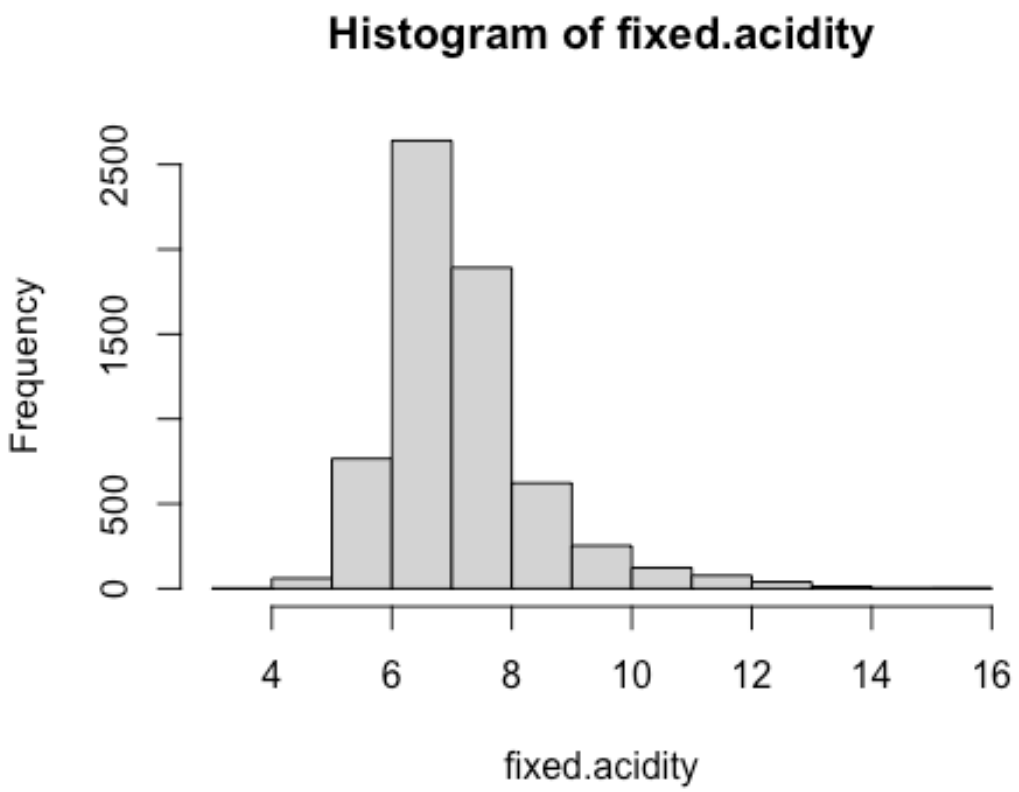
Mohamed Abdelrahman

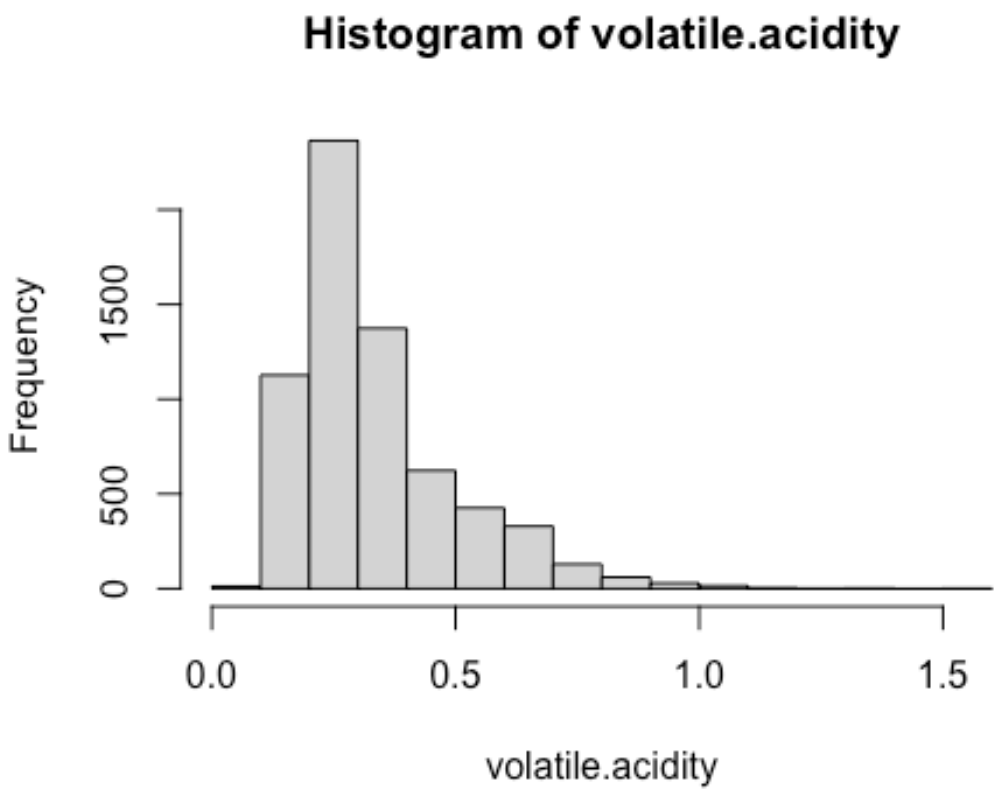
2023-12-01

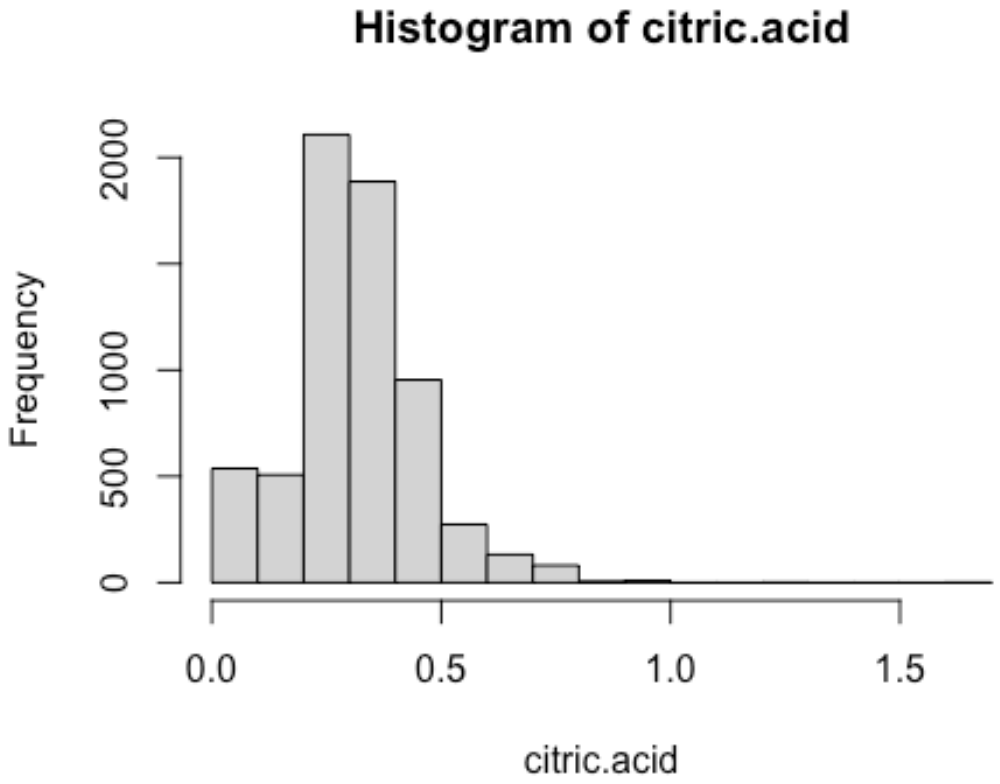
Data Processing:

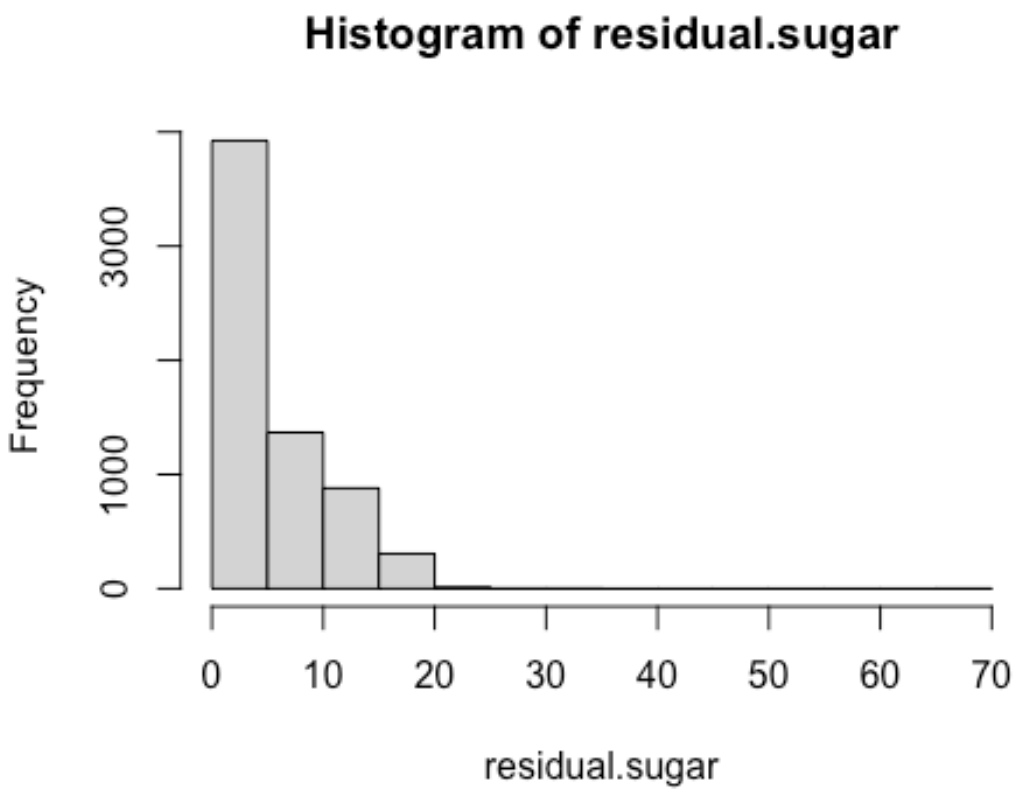
```
# Data Processing
set.seed(21)
WineDataDF <- as.data.frame(read.csv("wine-quality-white-and-red.csv", header
= TRUE))
WineDataDF$type <- ifelse(WineDataDF$type == "white", 1, 0)
# Plot histograms data
p <- ncol(WineDataDF)
for (i in 1:p){
  hist(x = as.numeric(WineDataDF[,i]), xlab = names(WineDataDF)[i], main = pa
ste("Histogram of", names(WineDataDF)[i], sep = " "))
}
```

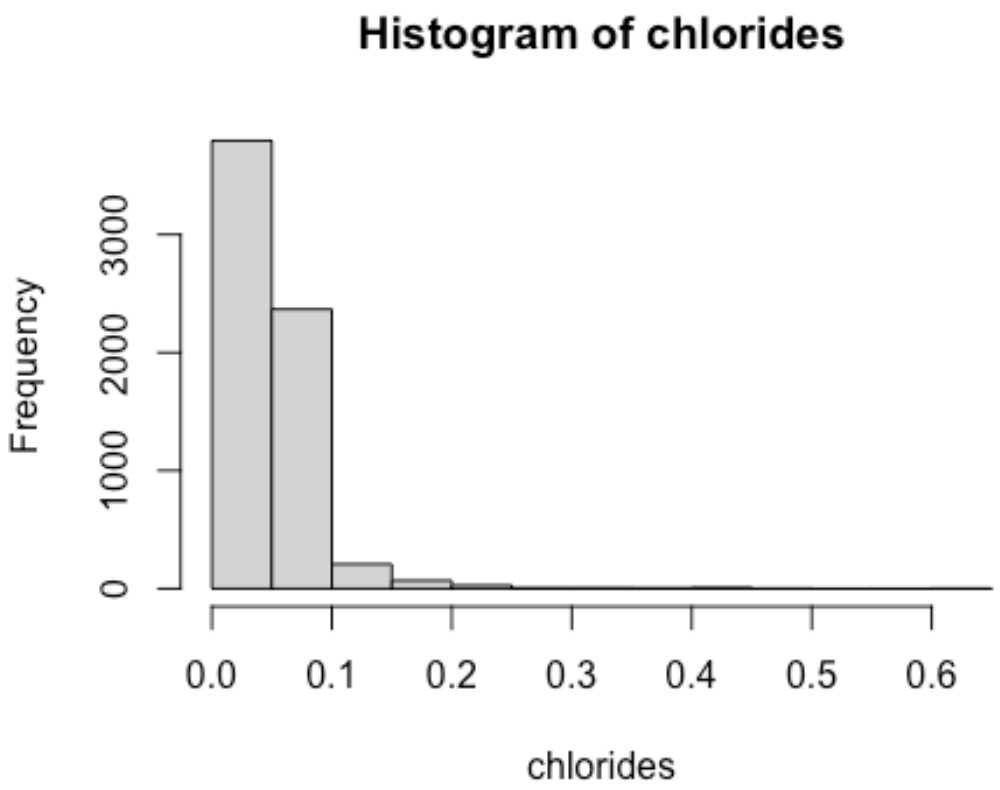

Histogram of type
Frequency
0
1000
3000
5000
0.0
0.2
0.4
0.6
0.8
1.0
type

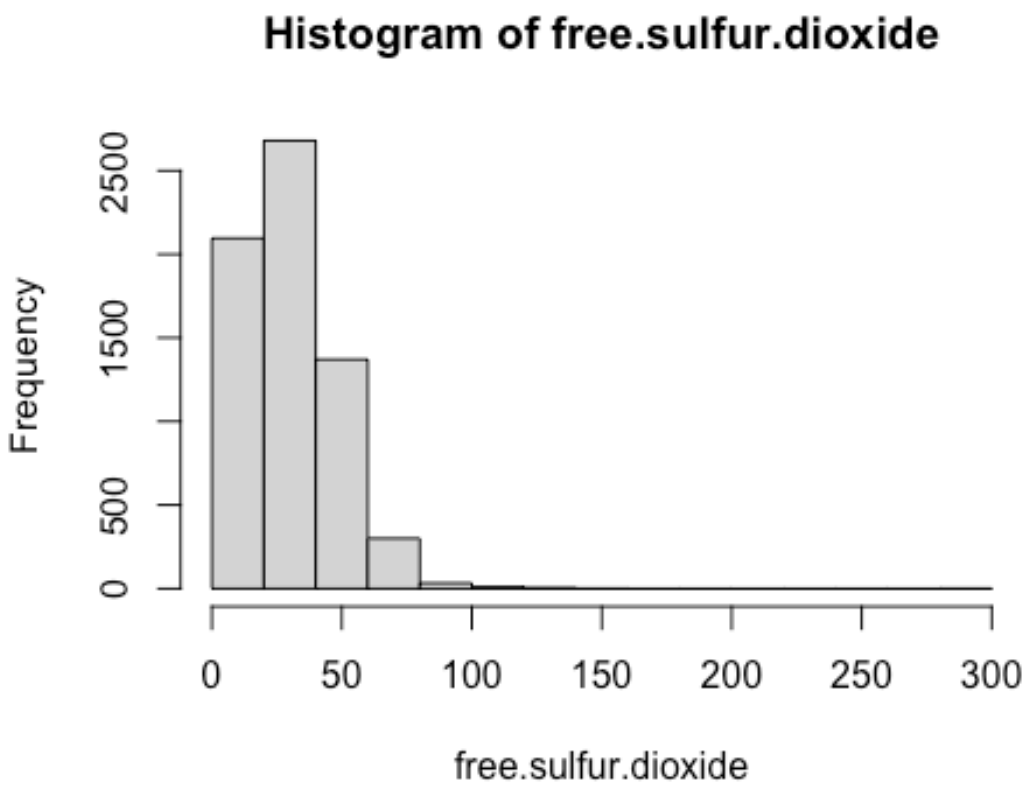

Histogram of fixed.acidity
Frequency
0
500
1500
2500
4
6
8
10
12
14
16
fixed.acidity

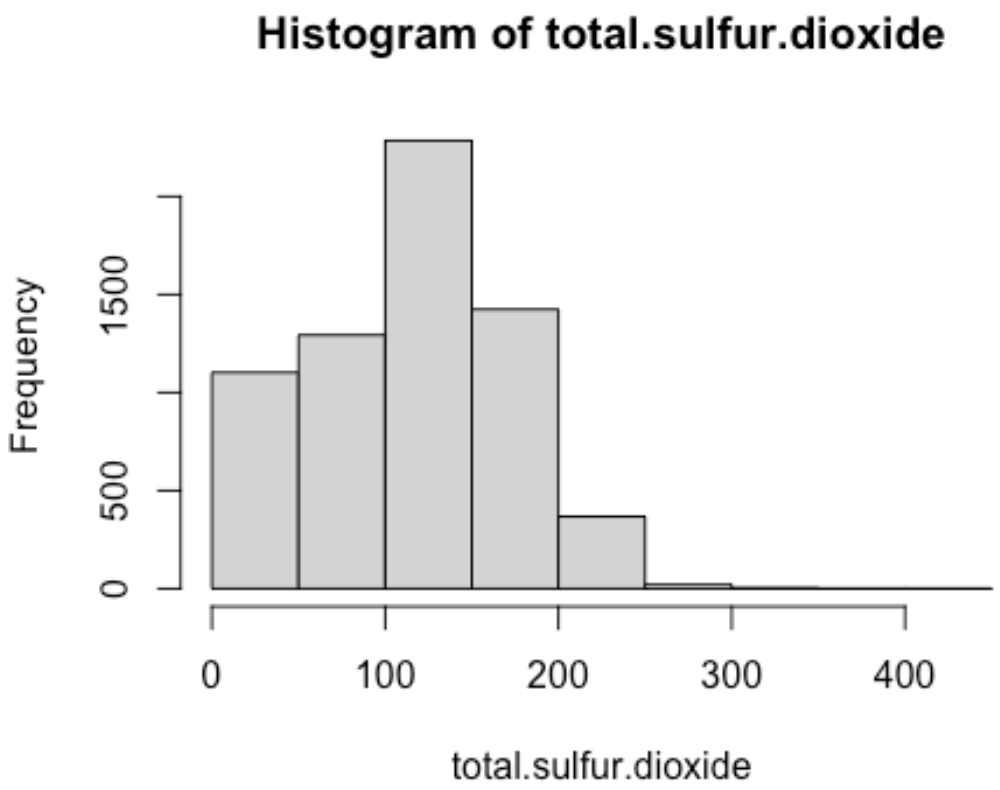

Histogram of volatile.acidity
Frequency
0
500
1500
0.0
0.5
1.0
1.5
volatile.acidity

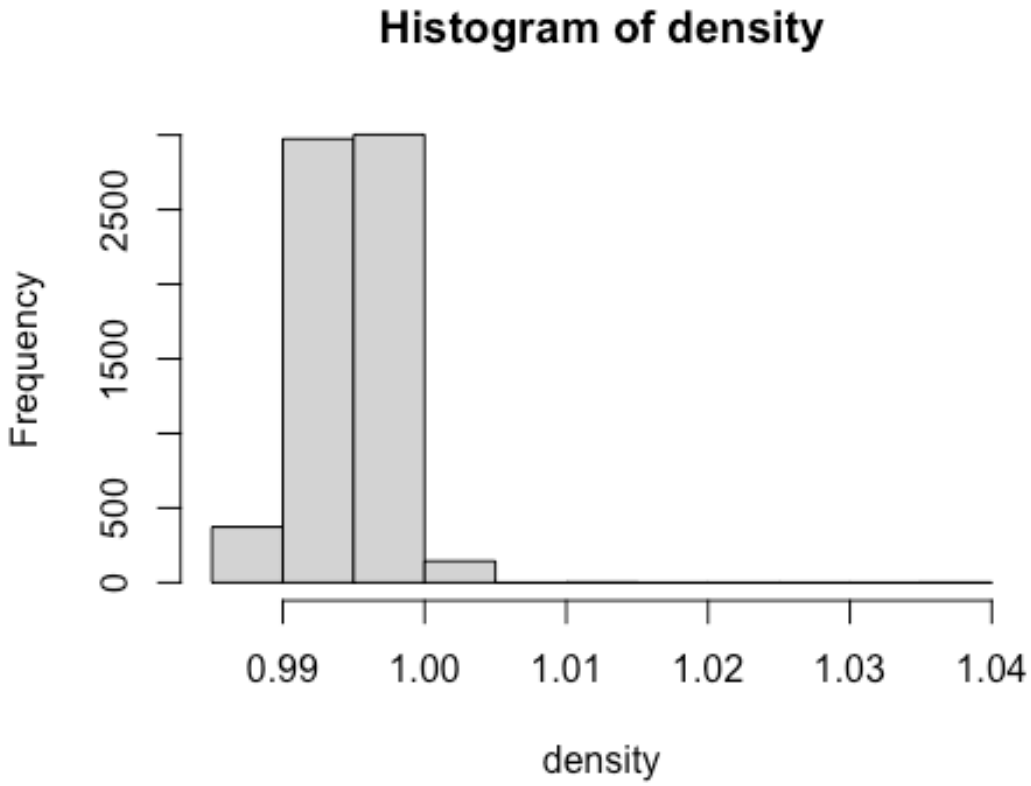

Histogram of citric.acid
Frequency
0
500
1000
2000
0.0
0.5
1.0
1.5
citric.acid

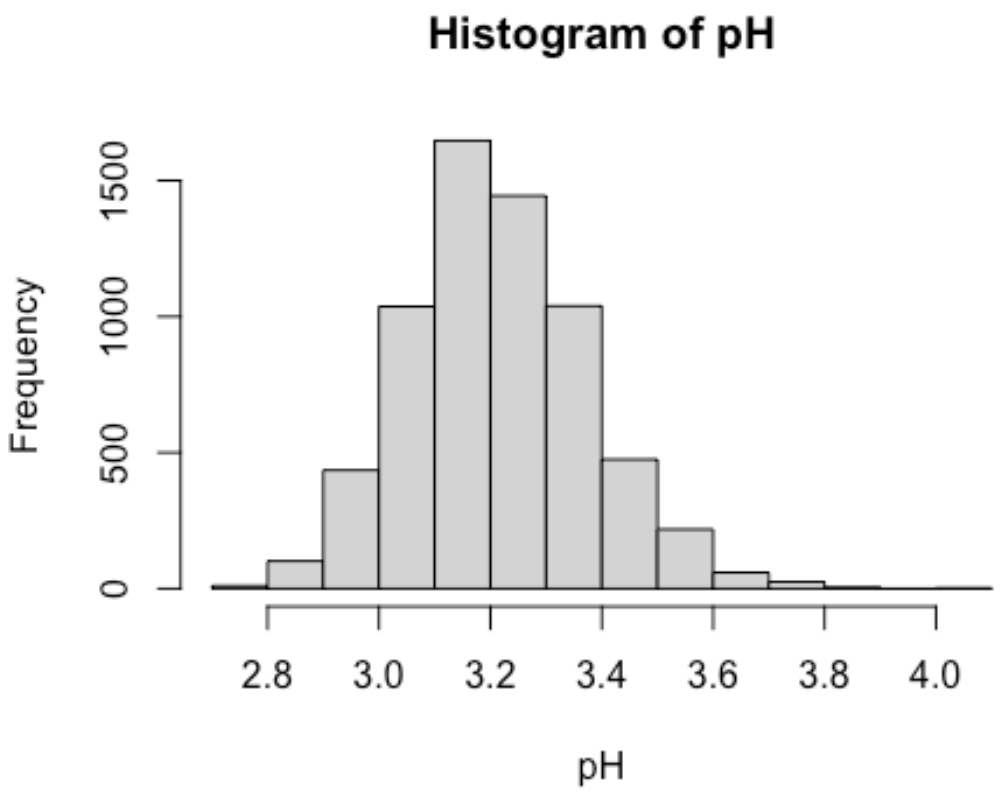

Histogram of residual.sugar
Frequency
0
1000
3000
0
10
20
30
40
50
60
70
residual.sugar

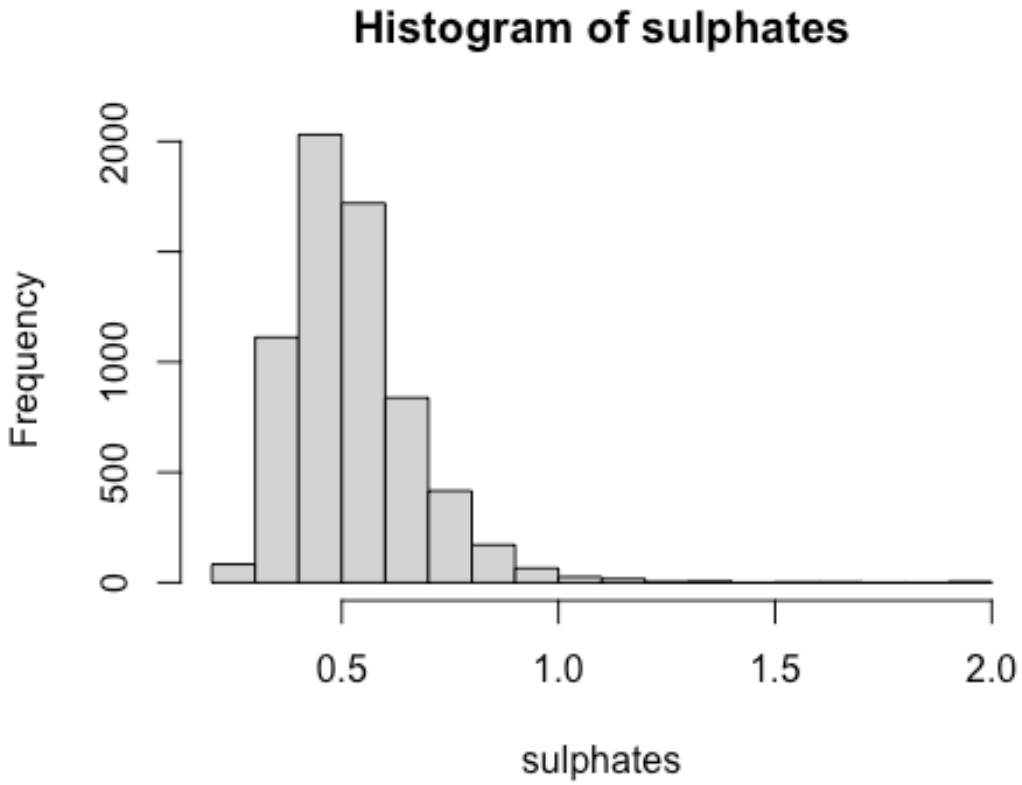

Histogram of chlorides
Frequency
0
1000
2000
3000
0.0
0.1
0.2
0.3
0.4
0.5
0.6
chlorides

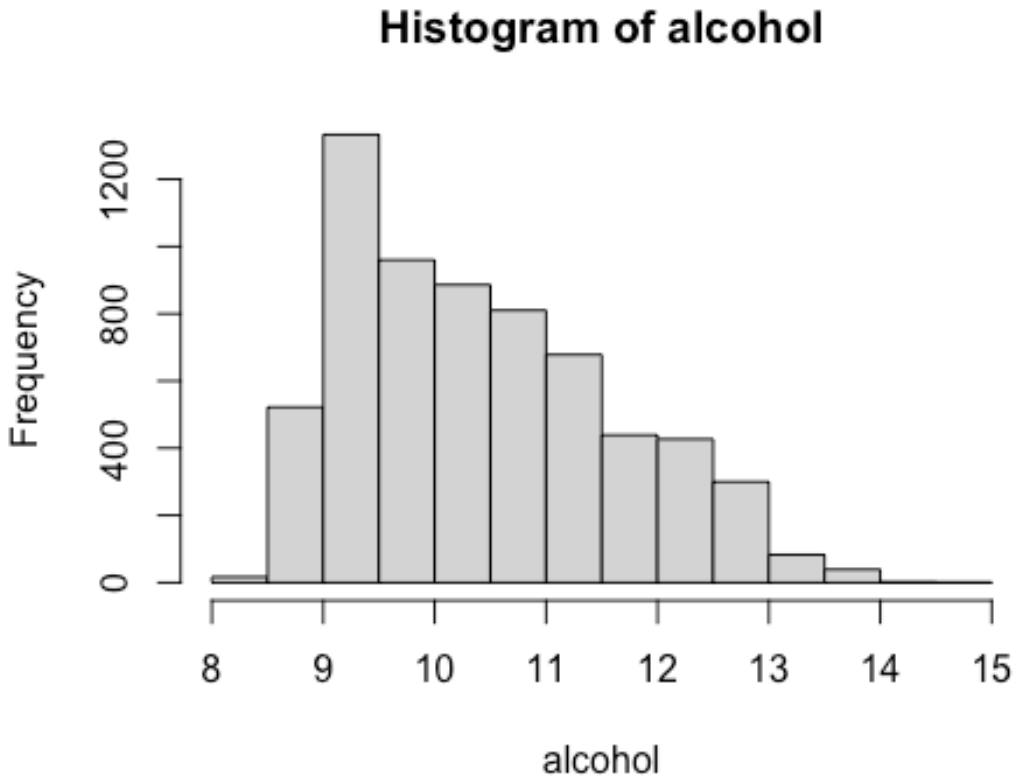

Histogram of free.sulfur.dioxide
Frequency
0
500
1500
2500
0
50
100
150
200
250
300
free.sulfur.dioxide

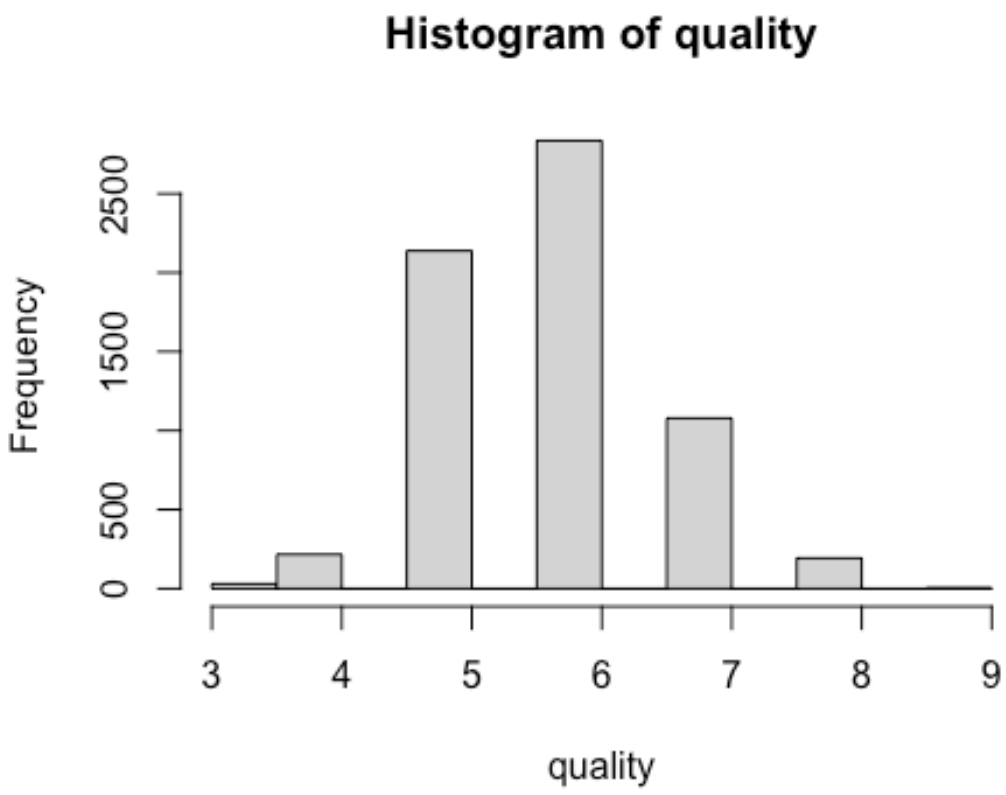

Histogram of total.sulfur.dioxide
Frequency
0
500
1500
0
100
200
300
400
total.sulfur.dioxide


Histogram of density
Frequency
0
500
1500
2500
0.99
1.00
1.01
1.02
1.03
1.04
density


Histogram of pH
Frequency
0
500
1000
1500
2.8
3.0
3.2
3.4
3.6
3.8
4.0
pH


Histogram of sulphates
Frequency
0
500
1000
2000
0.5
1.0
1.5
2.0
sulphates


Histogram of alcohol
Frequency
0
400
800
1200
8
9
10
11
12
13
14
15
alcohol




```r
table(WineDataDF$quality)
```

```
##
##    3    4    5    6    7    8    9
##   30  216 2138 2836 1079  193    5
```

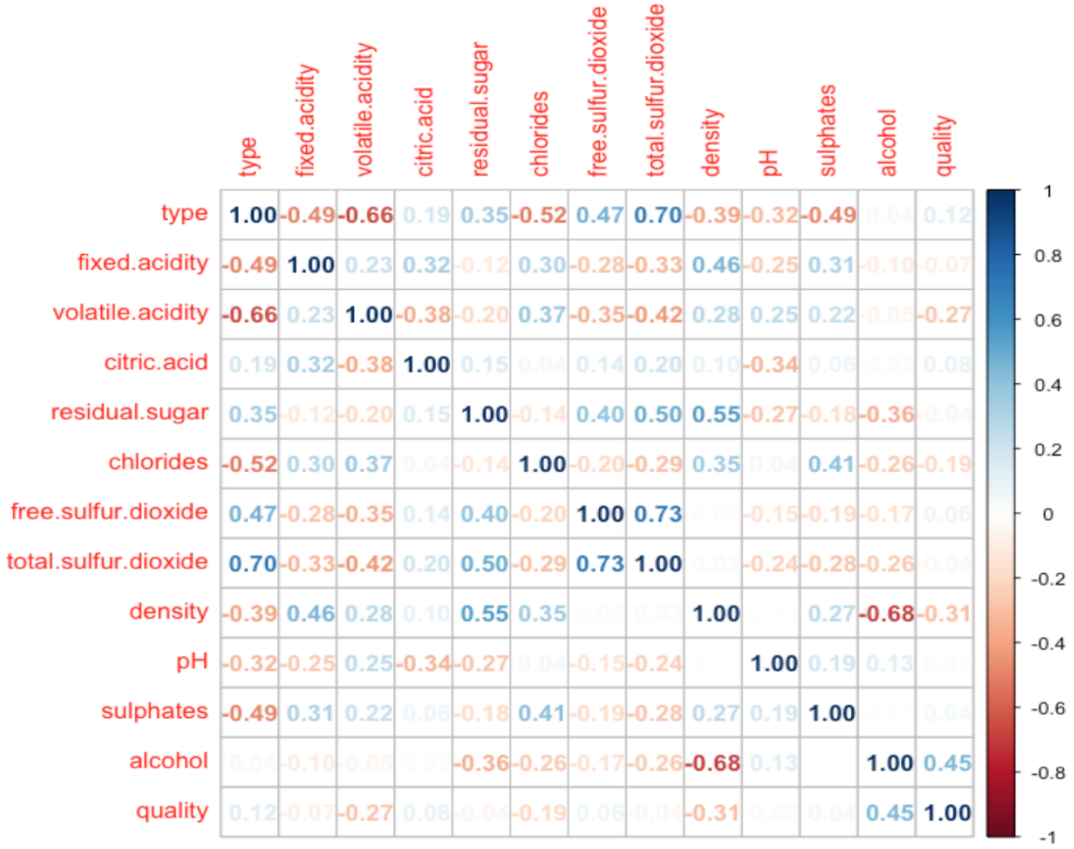
```r
# To avoid QDA or Naive Bayes breaking down, we merge the smallest classes to
the closest biggest class
WineDataDF$quality[WineDataDF$quality == 9] <- 8
WineDataDF$quality[WineDataDF$quality == 3] <- 4
table(WineDataDF$quality)
```

```
##
##    4    5    6    7    8
##  246 2138 2836 1079  198
```

```r
# A simple validation set approach will be used - split data to (~ 20%) test
and (~ 80%) train
n <- nrow(WineDataDF)
train_index <- sample(1:n, size = round(0.8*n))
train_data <- WineDataDF[train_index, ]
test_data <- WineDataDF[-train_index, ]
```

```r
# Analyze predictors & response correlation
library(corrplot)

## corrplot 0.92 loaded

train_corr <- cor(train_data)
train_corr

##                          type fixed.acidity volatile.acidity citric.aci
d
## type                  1.00000000   -0.48985788      -0.65734718  0.1887442
9
## fixed.acidity        -0.48985788    1.00000000       0.22519380  0.3211438
9
## volatile.acidity     -0.65734718    0.22519380       1.00000000 -0.3751503
7
## citric.acid           0.18874429    0.32114389      -0.37515037  1.0000000
0
## residual.sugar        0.35283396   -0.11503701      -0.19886955  0.1468076
0
## chlorides            -0.51597170    0.29752339       0.37024972  0.0448778
2
## free.sulfur.dioxide   0.47076318   -0.27625743      -0.35224435  0.1386651
7
## total.sulfur.dioxide  0.70343647   -0.33338922      -0.42168370  0.2002479
1
## density              -0.39130993    0.45952188       0.27503484  0.0993536
5
## pH                   -0.32416288   -0.25263527       0.25409292 -0.3351936
5
## sulphates            -0.48920011    0.31010426       0.22329332  0.0642915
8
## alcohol               0.03790065   -0.09905516      -0.04509854 -0.0172132
8
## quality               0.12360743   -0.07257874      -0.27205317  0.0840929
9
##                      residual.sugar   chlorides free.sulfur.dioxide
## type                      0.3528340 -0.51597170          0.47076318
## fixed.acidity            -0.1150370  0.29752339         -0.27625743
## volatile.acidity         -0.1988695  0.37024972         -0.35224435
## citric.acid               0.1468076  0.04487782          0.13866517
## residual.sugar            1.0000000 -0.13857389          0.39535633
## chlorides                -0.1385739  1.00000000         -0.20308285
## free.sulfur.dioxide       0.3953563 -0.20308285          1.00000000
## total.sulfur.dioxide      0.4959751 -0.29129213          0.72539500
## density                   0.5517208  0.35457023          0.01738796
## pH                       -0.2708072  0.04048648         -0.14855934
## sulphates                -0.1833223  0.41256205         -0.18919694
## alcohol                  -0.3583146 -0.25590068         -0.16775467
## quality                  -0.0429262 -0.19135030          0.06462266
```

```
##                       total.sulfur.dioxide      density           pH   sulp
hates
## type                           0.70343647 -0.391309928 -0.324162878 -0.489
20011
## fixed.acidity                 -0.33338922  0.459521875 -0.252635269  0.310
10426
## volatile.acidity              -0.42168370  0.275034845  0.254092920  0.223
29332
## citric.acid                    0.20024791  0.099353653 -0.335193654  0.064
29158
## residual.sugar                 0.49597514  0.551720817 -0.270807243 -0.183
32231
## chlorides                     -0.29129213  0.354570229  0.040486479  0.412
56205
## free.sulfur.dioxide            0.72539500  0.017387958 -0.148559338 -0.189
19694
## total.sulfur.dioxide           1.00000000  0.025612930 -0.237276229 -0.278
75549
## density                        0.02561293  1.000000000  0.006596032  0.265
35256
## pH                            -0.23727623  0.006596032  1.000000000  0.185
46996
## sulphates                     -0.27875549  0.265352555  0.185469964  1.000
00000
## alcohol                       -0.25810151 -0.684660827  0.125124356 -0.006
26715
## quality                       -0.03562948 -0.308911416  0.021830456  0.043
94295
##                          alcohol     quality
## type                  0.03790065  0.12360743
## fixed.acidity        -0.09905516 -0.07257874
## volatile.acidity     -0.04509854 -0.27205317
## citric.acid          -0.01721328  0.08409299
## residual.sugar       -0.35831457 -0.04292620
## chlorides            -0.25590068 -0.19135030
## free.sulfur.dioxide  -0.16775467  0.06462266
## total.sulfur.dioxide -0.25810151 -0.03562948
## density              -0.68466083 -0.30891142
## pH                    0.12512436  0.02183046
## sulphates            -0.00626715  0.04394295
## alcohol               1.00000000  0.44886864
## quality               0.44886864  1.00000000

corrplot(train_corr, method = "number")
```

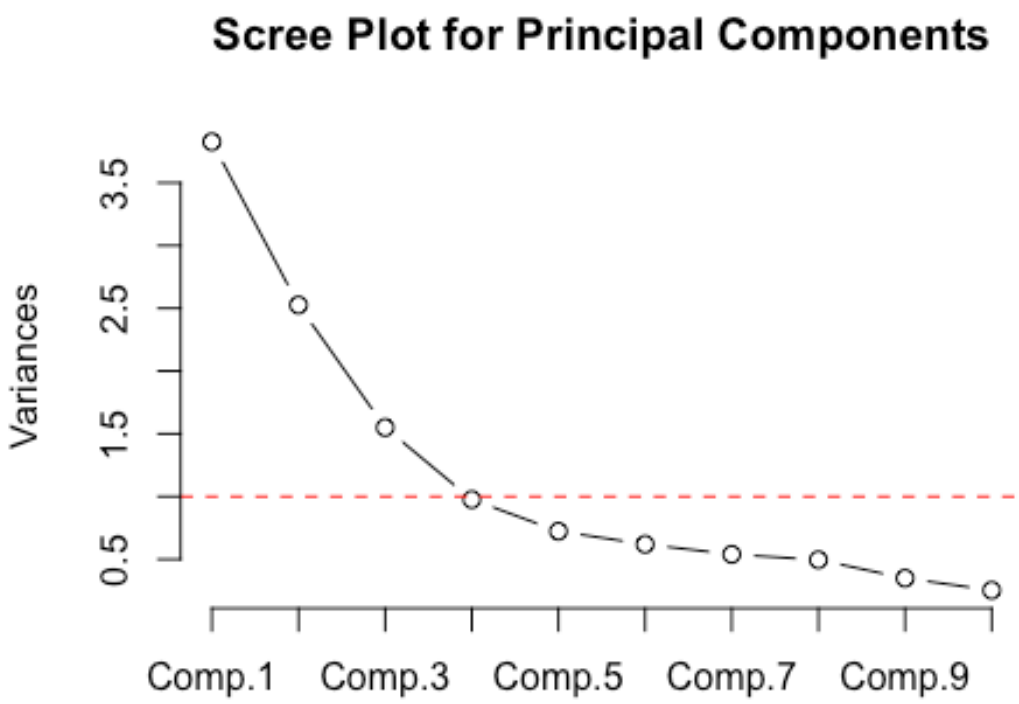



```
# find predictors correlated with quality
sort(abs(train_corr[,13]))
```

```
##                   pH total.sulfur.dioxide       residual.sugar
##           0.02183046           0.03562948           0.04292620
##            sulphates  free.sulfur.dioxide        fixed.acidity
##           0.04394295           0.06462266           0.07257874
##          citric.acid                 type            chlorides
##           0.08409299           0.12360743           0.19135030
##     volatile.acidity              density              alcohol
##           0.27205317           0.30891142           0.44886864
##              quality
##           1.00000000
```

```
# change quality to factor variable
train_data$quality <- as.factor(train_data$quality)
test_data$quality <- as.factor(test_data$quality)
y <- test_data$quality

# PCA analysis
xtrain <- train_data[,-13]
pca.results <- princomp(xtrain, cor = TRUE)
summary(pca.results)
```

```
## Importance of components:
##                           Comp.1    Comp.2    Comp.3     Comp.4     Comp.5
## Standard deviation     1.9562464 1.5894372 1.2449498 0.98782704 0.85038421
## Proportion of Variance 0.3189083 0.2105259 0.1291583 0.08131686 0.06026278
## Cumulative Proportion  0.3189083 0.5294342 0.6585925 0.73990939 0.80017216
##                            Comp.6     Comp.7     Comp.8     Comp.9   Comp.
10
## Standard deviation     0.78698676 0.73417950 0.70397298 0.59072624 0.50250
14
## Proportion of Variance 0.05161235 0.04491829 0.04129816 0.02907979 0.02104
23
## Cumulative Proportion  0.85178451 0.89670280 0.93800097 0.96708076 0.98812
31
##                           Comp.11     Comp.12
## Standard deviation     0.342843282 0.158056189
## Proportion of Variance 0.009795126 0.002081813
## Cumulative Proportion  0.997918187 1.000000000

# Scree plot - helps in choosing predictors
screeplot(pca.results, type = "l", main = "Scree Plot for Principal Component
s")
abline(1,0,col='red', lty = 2)
```



```r
pca.df <- as.data.frame(pca.results$scores)
pca.df$quality <- train_data$quality
# test PCs
xtest <- test_data[,-13]
test_pca.df <- as.data.frame(princomp(xtest, cor = TRUE)$scores)
```

Multinomial Logistic Regression Analysis:

```
library(nnet)
set.seed(21)
mlr.fit <- multinom(quality ~ volatile.acidity + alcohol + chlorides + fixed.
acidity + citric.acid + free.sulfur.dioxide, data = train_data, trace = FALSE
)
summary(mlr.fit)

## Call:
## multinom(formula = quality ~ volatile.acidity + alcohol + chlorides +
##     fixed.acidity + citric.acid + free.sulfur.dioxide, data = train_data,
##     trace = FALSE)
##
## Coefficients:
##   (Intercept) volatile.acidity    alcohol chlorides fixed.acidity citric.a
cid
## 5    4.479142        -1.878776 -0.2871867  8.814030    0.07287285  -0.7679
184
## 6   -2.824086        -5.282734  0.5170752 10.787625    0.13368868  -1.2325
637
## 7  -10.854094        -7.580875  1.1534948  6.121157    0.28196095  -1.6905
797
## 8  -15.251534        -7.187786  1.4024344  6.749300    0.15091297  -1.4121
629
##   free.sulfur.dioxide
## 5          0.01920671
## 6          0.02469102
## 7          0.02698563
## 8          0.04166958
##
## Std. Errors:
##   (Intercept) volatile.acidity    alcohol chlorides fixed.acidity citric.a
cid
## 5    1.038905        0.4369045 0.08056046  1.852892    0.06788256   0.5716
428
## 6    1.027815        0.4654825 0.07837330  1.815503    0.06804380   0.5797
604
## 7    1.127219        0.5856970 0.08485509  2.322204    0.07492790   0.6680
688
## 8    1.563030        0.9633748 0.11156008  2.823514    0.11021438   0.9786
236
##   free.sulfur.dioxide
## 5         0.005694152
## 6         0.005691278
## 7         0.006084781
## 8         0.007102047
##
## Residual Deviance: 11178.34
## AIC: 11234.34
```

```r
mlr_pred <- predict(mlr.fit, newdata = test_data)
mlr_test_error <- mean(y != mlr_pred)
mlr_test_error
```

```
## [1] 0.5011547
```

```r
pca_mlr.fit <- multinom(quality ~ Comp.1 + Comp.2 + Comp.3 + Comp.4, data = p
ca.df, trace = FALSE)
summary(pca_mlr.fit)
```

```
## Call:
## multinom(formula = quality ~ Comp.1 + Comp.2 + Comp.3 + Comp.4,
##     data = pca.df, trace = FALSE)
##
## Coefficients:
##   (Intercept)     Comp.1     Comp.2      Comp.3     Comp.4
## 5   2.2057442 0.03525379 -0.3012897 -0.03199728 -0.3360473
## 6   2.6217161 0.16900662  0.0228645  0.27984879 -0.6284936
## 7   1.3634150 0.20154050  0.3815033  0.58612301 -0.8248439
## 8  -0.6106737 0.35994750  0.5207432  0.55101675 -1.0140480
##
## Std. Errors:
##   (Intercept)     Comp.1     Comp.2     Comp.3     Comp.4
## 5  0.08811522 0.03663036 0.05373657 0.06024253 0.08541830
## 6  0.08626056 0.03699018 0.05295165 0.06010885 0.08507284
## 7  0.09459730 0.04224493 0.05748442 0.06633190 0.09082155
## 8  0.14902395 0.07352139 0.08057170 0.09913657 0.11914647
##
## Residual Deviance: 11828.48
## AIC: 11868.48
```

```r
pca_mlr_pred <- predict(pca_mlr.fit, newdata = test_pca.df)
pca_mlr_test_error <- mean(y != pca_mlr_pred)
pca_mlr_test_error
```

```
## [1] 0.5327175
```

Linear Discriminant Analysis, Quadratic Discriminant Analysis, and Naive Bayes Analyses:

```
library(MASS)
library(e1071)
library(klaR)
options(warn = -1)
set.seed(21)
# LDA
lda.fit <- lda(quality ~ volatile.acidity + alcohol + chlorides + fixed.acidi
ty + citric.acid + free.sulfur.dioxide, data = train_data)
lda_pred <- predict(lda.fit, newdata = test_data)
lda_test_error <- mean(y != lda_pred$class)
lda_test_error
```

```
## [1] 0.5019246
```

```
pca_lda.fit <- lda(quality ~ Comp.1 + Comp.2 + Comp.3 + Comp.4, data = pca.df
)
pca_lda_pred <- predict(pca_lda.fit, newdata = test_pca.df)
pca_lda_test_error <- mean(y != pca_lda_pred$class)
pca_lda_test_error
```

```
## [1] 0.5342571
```

```
# QDA
qda.fit <- qda(quality ~ volatile.acidity + alcohol + chlorides + fixed.acidi
ty + citric.acid + free.sulfur.dioxide, data = train_data)
qda_pred <- predict(qda.fit, newdata = test_data)
qda_test_error <- mean(y != qda_pred$class)
qda_test_error
```

```
## [1] 0.5273287
```

```
pca_qda.fit <- qda(quality ~ Comp.1 + Comp.2 + Comp.3 + Comp.4, data = pca.df
)
pca_qda_pred <- predict(pca_qda.fit, newdata = test_pca.df)
pca_qda_test_error <- mean(y != pca_qda_pred$class)
pca_qda_test_error
```

```
## [1] 0.5288684
```

```
# Naive Bayes
nb.fit <- naiveBayes(quality ~ volatile.acidity + alcohol + chlorides + fixed
.acidity + citric.acid + free.sulfur.dioxide, data = train_data)
nb_pred <- predict(nb.fit, newdata = test_data)
nb_test_error <- mean(y != nb_pred)
nb_test_error
```

```
## [1] 0.5419554
```

```r
knb.fit <- NaiveBayes(quality ~ volatile.acidity + alcohol + chlorides + fixe
d.acidity + citric.acid + free.sulfur.dioxide, data = train_data, usekernel =
TRUE)
knb_pred <- predict(knb.fit, newdata = test_data)
knb_test_error <- mean(y != knb_pred$class)
knb_test_error
```

```
## [1] 0.4996151
```

```r
pca_nb.fit <- naiveBayes(quality ~ Comp.1 + Comp.2 + Comp.3 + Comp.4, data =
pca.df)
pca_nb_pred <- predict(pca_nb.fit, newdata = test_pca.df)
pca_nb_test_error <- mean(y != pca_nb_pred)
pca_nb_test_error
```

```
## [1] 0.5488838
```

```r
kpca_nb.fit <- NaiveBayes(quality ~ Comp.1 + Comp.2 + Comp.3 + Comp.4, data =
pca.df, usekernel = TRUE)
kpca_nb_pred <- predict(kpca_nb.fit, newdata = test_pca.df)
kpca_nb_test_error <- mean(y != kpca_nb_pred$class)
kpca_nb_test_error
```

```
## [1] 0.5450346
```

KNN Analysis:

```
library(class)
set.seed(21)
N <- 100
knn_test_errors <- rep(0, N)
knn_pca_test_errors <- rep(0, N)
Xtr <- as.data.frame(cbind(xtrain$volatile.acidity, xtrain$alcohol, xtrain$ch
lorides, xtrain$fixed.acidity, xtrain$citric.acid, xtrain$free.sulfur.dioxide
))
Xte <- as.data.frame(cbind(xtest$volatile.acidity, xtest$alcohol, xtest$chlor
ides, xtest$fixed.acidity, xtest$citric.acid, xtest$free.sulfur.dioxide))
pcatrain <- pca.df[,1:4]
pcatest <- test_pca.df[,1:4]
for(i in 1:N){
  knn.pred <- knn(Xtr, Xte, train_data$quality, k = i)
  knn_test_errors[i] <- mean(y != knn.pred)
  pca_knn.pred <- knn(pcatrain, pcatest, pca.df$quality, k = i)
  knn_pca_test_errors[i] <- mean(y != pca_knn.pred)
}
min(knn_test_errors)

## [1] 0.4272517

k1 <- which(knn_test_errors == min(knn_test_errors))
k1

## [1] 1

min(knn_pca_test_errors)

## [1] 0.4603541

k2 <- which(knn_pca_test_errors == min(knn_pca_test_errors))
k2

## [1] 1
```

Support Vector Machine Analysis:

```
library(e1071)
set.seed(21)
# fit best model
lin_tune.out <- tune(svm , quality ~ volatile.acidity + alcohol + chlorides +
fixed.acidity + citric.acid + free.sulfur.dioxide, data = train_data , kernel
= "linear",
ranges = list(cost = c(0.001 , 0.01, 0.1, 1, 5, 10, 100)))
summary(lin_tune.out)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##   0.1
##
## - best performance: 0.4763376
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-03 0.5101975 0.02050072
## 2 1e-02 0.4771050 0.02241638
## 3 1e-01 0.4763376 0.02152701
## 4 1e+00 0.4767215 0.02091620
## 5 5e+00 0.4769142 0.02159470
## 6 1e+01 0.4769138 0.02098592
## 7 1e+02 0.4767219 0.02162219

pca_lin_tune.out <- tune(svm , quality ~ Comp.1 + Comp.2 + Comp.3 + Comp.4, d
ata = pca.df, kernel = "linear",
ranges = list(cost = c(0.001 , 0.01, 0.1, 1, 5, 10, 100)))
summary(pca_lin_tune.out)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost
##  0.01
##
## - best performance: 0.5065403
##
## - Detailed performance results:
##    cost     error dispersion
## 1 1e-03 0.5384745 0.02012961
```

```
## 2 1e-02 0.5065403 0.02618149
## 3 1e-01 0.5073107 0.02757305
## 4 1e+00 0.5071180 0.02716364
## 5 5e+00 0.5071180 0.02756905
## 6 1e+01 0.5073103 0.02756531
## 7 1e+02 0.5069257 0.02751163
```

```
rad_tune.out <- tune(svm , quality ~ volatile.acidity + alcohol + chlorides +
fixed.acidity + citric.acid + free.sulfur.dioxide, data = train_data, kernel
= "radial",ranges = list(cost = c(0.1 , 1, 10, 100, 1000) ,gamma = c(0.5, 1,
2, 3, 4)))
summary(rad_tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##    10     4
##
## - best performance: 0.3772554
##
## - Detailed performance results:
##     cost gamma     error dispersion
## 1  1e-01   0.5 0.4501775 0.01657389
## 2  1e+00   0.5 0.4268979 0.02262291
## 3  1e+01   0.5 0.4178439 0.02771042
## 4  1e+02   0.5 0.4291915 0.02373338
## 5  1e+03   0.5 0.4301601 0.01644398
## 6  1e-01   1.0 0.4580636 0.02160149
## 7  1e+00   1.0 0.4180447 0.02129251
## 8  1e+01   1.0 0.4124574 0.02145121
## 9  1e+02   1.0 0.4164999 0.01994827
## 10 1e+03   1.0 0.4003387 0.02257145
## 11 1e-01   2.0 0.5215459 0.01860857
## 12 1e+00   2.0 0.4089988 0.01826976
## 13 1e+01   2.0 0.3991852 0.02191056
## 14 1e+02   2.0 0.3928353 0.01793947
## 15 1e+03   2.0 0.3947580 0.01962204
## 16 1e-01   3.0 0.5513669 0.02051187
## 17 1e+00   3.0 0.4009204 0.01819584
## 18 1e+01   3.0 0.3851412 0.01755295
## 19 1e+02   3.0 0.3887972 0.01925630
## 20 1e+03   3.0 0.3893745 0.01933930
## 21 1e-01   4.0 0.5594472 0.02124417
## 22 1e+00   4.0 0.3897599 0.02289102
## 23 1e+01   4.0 0.3772554 0.01584376
```

```
## 24 1e+02   4.0 0.3793708 0.01778701
## 25 1e+03   4.0 0.3795631 0.01822466
```

```
pca_rad_tune.out <- tune(svm , quality ~ Comp.1 + Comp.2 + Comp.3 + Comp.4, d
ata = pca.df, kernel = "radial",ranges = list(cost = c(0.1 , 1, 10, 100, 1000
) ,gamma = c(0.5, 1, 2, 3, 4)))
summary(pca_rad_tune.out)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost gamma
##  1000     3
##
## - best performance: 0.4090014
##
## - Detailed performance results:
##     cost gamma     error  dispersion
## 1  1e-01   0.5 0.4909612 0.017551784
## 2  1e+00   0.5 0.4746087 0.013687116
## 3  1e+01   0.5 0.4661442 0.022363770
## 4  1e+02   0.5 0.4640259 0.019347258
## 5  1e+03   0.5 0.4578665 0.017894521
## 6  1e-01   1.0 0.4894216 0.016029128
## 7  1e+00   1.0 0.4638365 0.020224062
## 8  1e+01   1.0 0.4588343 0.020135447
## 9  1e+02   1.0 0.4559452 0.022801945
## 10 1e+03   1.0 0.4445932 0.010965402
## 11 1e-01   2.0 0.4932703 0.018253246
## 12 1e+00   2.0 0.4594116 0.022861578
## 13 1e+01   2.0 0.4449774 0.016043066
## 14 1e+02   2.0 0.4388247 0.013529759
## 15 1e+03   2.0 0.4247743 0.019734682
## 16 1e-01   3.0 0.5046209 0.020371417
## 17 1e+00   3.0 0.4515240 0.023832479
## 18 1e+01   3.0 0.4444016 0.009919231
## 19 1e+02   3.0 0.4268938 0.018737911
## 20 1e+03   3.0 0.4090014 0.017512459
## 21 1e-01   4.0 0.5200100 0.020347161
## 22 1e+00   4.0 0.4411327 0.017691858
## 23 1e+01   4.0 0.4380554 0.013406354
## 24 1e+02   4.0 0.4136238 0.014155803
## 25 1e+03   4.0 0.4091971 0.012959329
```

```
# predictions
svml.pred <- predict(lin_tune.out$best.model, newdata=test_data)
```

```r
svml_test_error <- mean(y != svml.pred)
svml_test_error
```

```
## [1] 0.4934565
```

```r
pca_svml.pred <- predict(pca_lin_tune.out$best.model, newdata= test_pca.df)
pca_svml_test_error <- mean(y != pca_svml.pred)
pca_svml_test_error
```

```
## [1] 0.5327175
```

```r
svmr.pred <- predict(rad_tune.out$best.model, newdata=test_data)
svmr_test_error <- mean(y != svmr.pred)
svmr_test_error
```

```
## [1] 0.3918399
```

```r
pca_svmr.pred <- predict(pca_rad_tune.out$best.model, newdata=test_pca.df)
pca_svmr_test_error <- mean(y != pca_svmr.pred)
pca_svmr_test_error
```

```
## [1] 0.530408
```