Team 3

Class Project 2

A Case Study in Data Mining

12.20.2023

WINE QUALITY

Team Members :

Winona Patrick
Simranjit Singh
Li Temeran
Beyzanur Tokar
Zhenya Venglinski

Table of Contents

PART I. Introduction

Wine, a beverage celebrated for its rich history and cultural significance, is enjoyed by connoisseurs and casual drinkers alike across the globe. The quality of wine is not just a matter of taste, but also a pivotal factor in the wine industry's marketability and economic success. Understanding and predicting wine quality has, therefore, become a crucial task for winemakers, sommeliers, and wine enthusiasts.

Traditionally, wine quality is assessed by experts through sensory evaluation, considering aspects like taste, aroma, and color. However, this method is subjective and can vary widely between individuals. With advancements in science and technology, it has become apparent that the quality of wine is profoundly influenced by its physicochemical properties, such as acidity, alcohol content, pH levels, and sugar concentration, among others. These properties are measurable, offering a more objective approach to determining wine quality.

The challenge lies in effectively predicting the quality of wine based on these measurable physicochemical properties. This task not only requires a comprehensive understanding of the relationship between these properties and the perceived quality of wine but also necessitates a methodical approach to analyze and interpret this complex data. In recent years, data analysis and machine learning have emerged as powerful tools in addressing this challenge. By applying these techniques, we can analyze extensive datasets of wine properties and develop models that predict wine quality with high accuracy. Such models have the potential to revolutionize the wine industry, offering a more standardized and efficient way of assessing wine quality.

In this study, we focus on the research question, “Can we predict wine quality using physicochemical properties?”. To do this, we will be employing five different algorithms to predict wine quality using physicochemical properties. Each algorithm, with its unique approach to data analysis, offers a perspective in understanding and predicting the complexities of wine quality. We want to provide a reliable, objective, and scalable solution to assess wine quality, which could potentially benefit winemakers, marketers, and consumers by ensuring quality and consistency in wine production and consumption.

Part II. Data

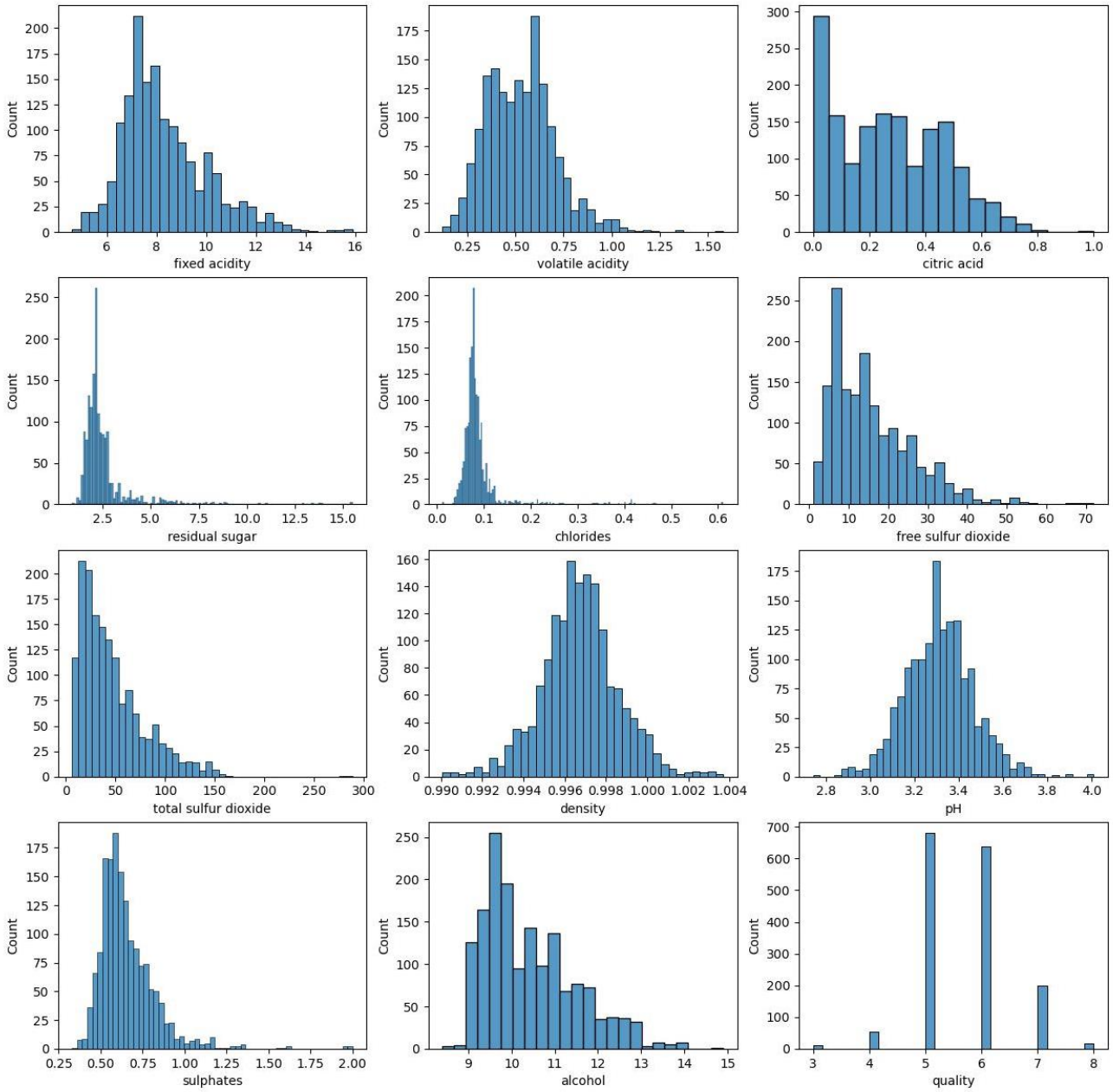
Data Discussion

Our team chose to work with the red wine data found in a wine quality dataset at the UC Irvine Machine Learning Repository (https://archive.ics.uci.edu/dataset/186/wine+quality). The dataset contains 1,599 records with 11 variables and no missing values. Ten of these variables are physiochemical properties of the specific wine which include Fixed Acidity, Volatile Acidity, Citric Acid, Residual Sugar, Chlorides, Free Sulfur Dioxide, Total Sulfur Dioxide, Density, pH, Sulphates, and Alcohol Content. These are independent quantitative variables that will be used as input for data analysis. What we are trying to determine is the final variable or label, Quality, a dependent categorical variable. Summary statistics for each variable are below.

Value	Fixed Acidity	Volatile Acidity	Citric Acid	Residual Sugar	Chlorides	Free Sulfur Dioxide
mean	8.32	0.53	0.27	2.54	0.09	15.87
std	1.74	0.18	0.19	1.41	0.05	10.46
min	4.6	0.12	0	0.9	0.01	1
25%	7.1	0.39	0.09	1.9	0.07	7
50%	7.9	0.52	0.26	2.2	0.08	14
75%	9.2	0.64	0.42	2.6	0.09	21
max	15.9	1.58	1	15.5	0.61	72

Value	Total Sulfur Dioxide	Density	pH	Sulphates	Alcohol	Quality
mean	46.47	1	3.31	0.66	10.42	5.64
std	32.9	0	0.15	0.17	1.07	0.81
min	6	0.99	2.74	0.33	8.4	3
25%	22	1	3.21	0.55	9.5	5
50%	38	1	3.31	0.62	10.2	6
75%	62	1	3.4	0.73	11.1	6
max	289	1	4.01	2	14.9	8

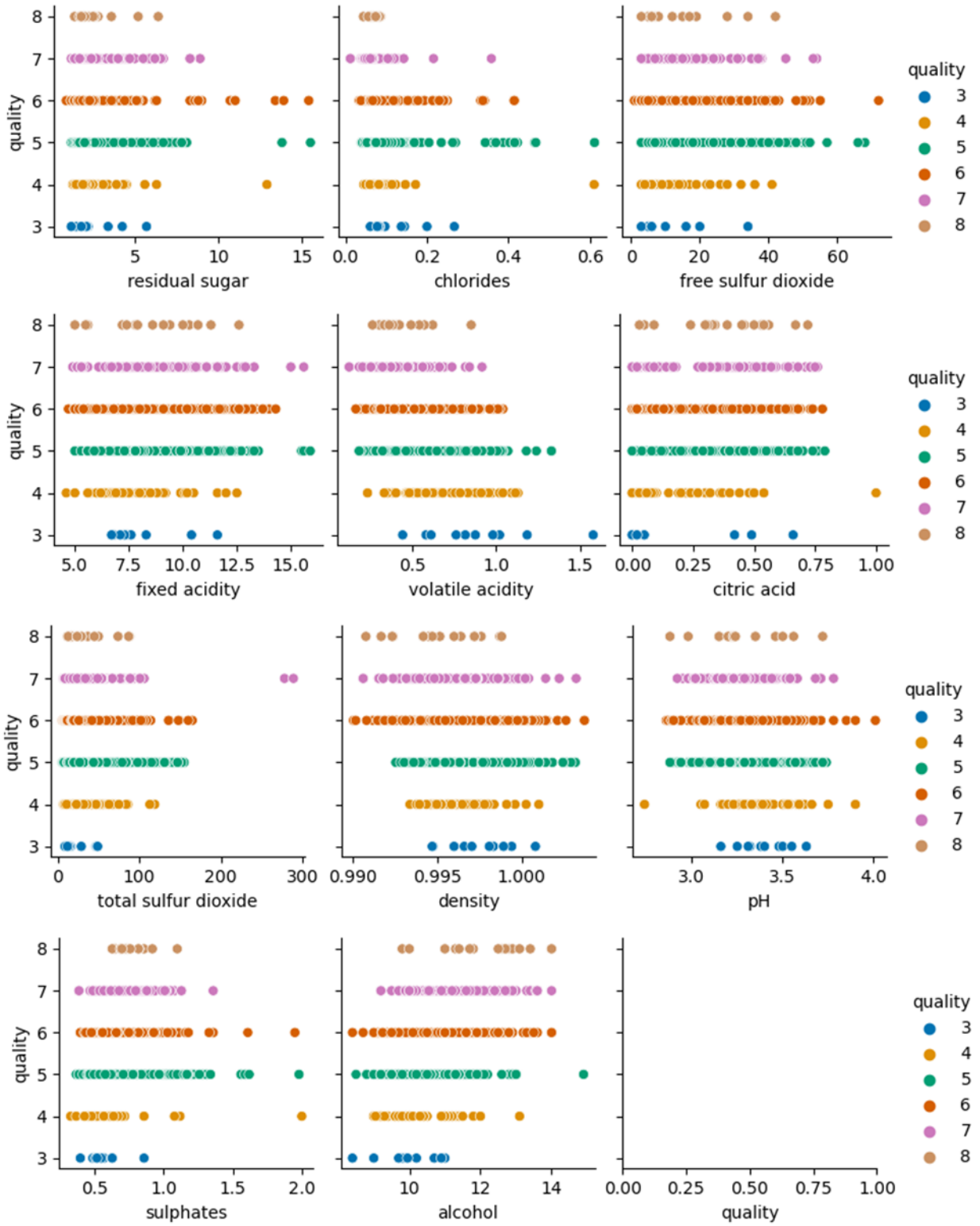
Histograms show the distributions of each variable. Several distributions are fairly close to a Normal curve (Fixed Acidity, Density, pH, and Quality). If outliers were removed, Volatile Acidity and Sulphates would also be fairly close to a Normal curve. All other variables are skewed right.



The table below shows correlation values between the various physiochemical properties and quality of the wine. Fixed Acidity, Citric Acid, Residual Sugar, Sulphates, and Alcohol Content all have positive correlation scores so as their values increase, the quality score of the wine increases. Volatile Acidity, Chlorides, Free Sulfur Dioxide, Total Sulfur Dioxide, Density, and pH all have negative correlation scores, so as their values increase, the quality of the wine decreases. Residual Sugar, Free Sulfur Dioxide, and pH have a low effect on wine quality since the absolute value of their scores are below 0.1. The top four variables that have an effect on wine quality are Citric Acid, Sulphates, Volatile Acidity, and Alcohol Content since the absolute value of their scores are above 0.2.

Physiochemical Property	Correlation to Wine Quality	Correlation (Abs Value)
Fixed Acidity	0.12	0.12
Volatile Acidity	-0.39	0.39
Citric Acid	0.23	0.23
Residual Sugar	0.01	0.01
Chlorides	-0.13	0.13
Free Sulfur Dioxide	-0.05	0.05
Total Sulfur Dioxide	-0.19	0.19
Density	-0.17	0.17
pH	-0.06	0.06
Sulphates	0.25	0.25
Alcohol Content	0.48	0.48

Pair plot showing distributions of variable distributions across quality scores.



Python Code Block

```
# Import packages
import os
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

#Read csv files
df = pd.read_csv('winequality-red.csv')

#Gets information on dataframe
#There are 1599 records with 12 attributes.
#Quality is an integer, and all others are floats
df.info()

#Check for missing values
#There are none
df.isnull() .sum()

#Generate summary statistics using describe function
df_stat = df.describe().drop(["count"]).round(3)

#Generate .csv file to add to report
df_stat.to_csv('Red_Summary_Stats.csv')

#Generate histograms to observe distributions
fig, axes = plt.subplots(nrows = 4, ncols = 3)    # axes is 2d array (3x3)
axes = axes.flatten()          # Convert axes to 1d array of length 9
fig.set_size_inches(15, 15)

for ax, col in zip(axes, df.columns):
  sns.histplot(df[col], ax = ax)

#Save figure to add to report
plt.savefig('Variable_Distributions.jpeg')

#Create pair plots and save figures to add to report
p1 = sns.pairplot(df, hue='quality', palette='colorblind',
             x_vars=('fixed acidity', 'volatile acidity', 'citric acid'),
             y_vars='quality')
plt.savefig('Pair_Plot1')

p2 = sns.pairplot(df, hue='quality', palette='colorblind',
             x_vars=('residual sugar', 'chlorides', 'free sulfur dioxide'),
             y_vars='quality')
plt.savefig('Pair_Plot2')

p3 = sns.pairplot(df, hue='quality', palette='colorblind',
             x_vars=('total sulfur dioxide', 'density', 'pH'),
             y_vars='quality')
plt.savefig('Pair_Plot3')

p4 = sns.pairplot(df, hue='quality', palette='colorblind',
             x_vars=('sulphates', 'alcohol', 'quality'),
```

```
            y_vars='quality')
plt.savefig('Pair_Plot4')

#Create correlation table and save as .csv
corr = df.corrwith(df['quality']).round(2)
print(corr)
corr.to_csv('Quality_Correlation.csv')
```

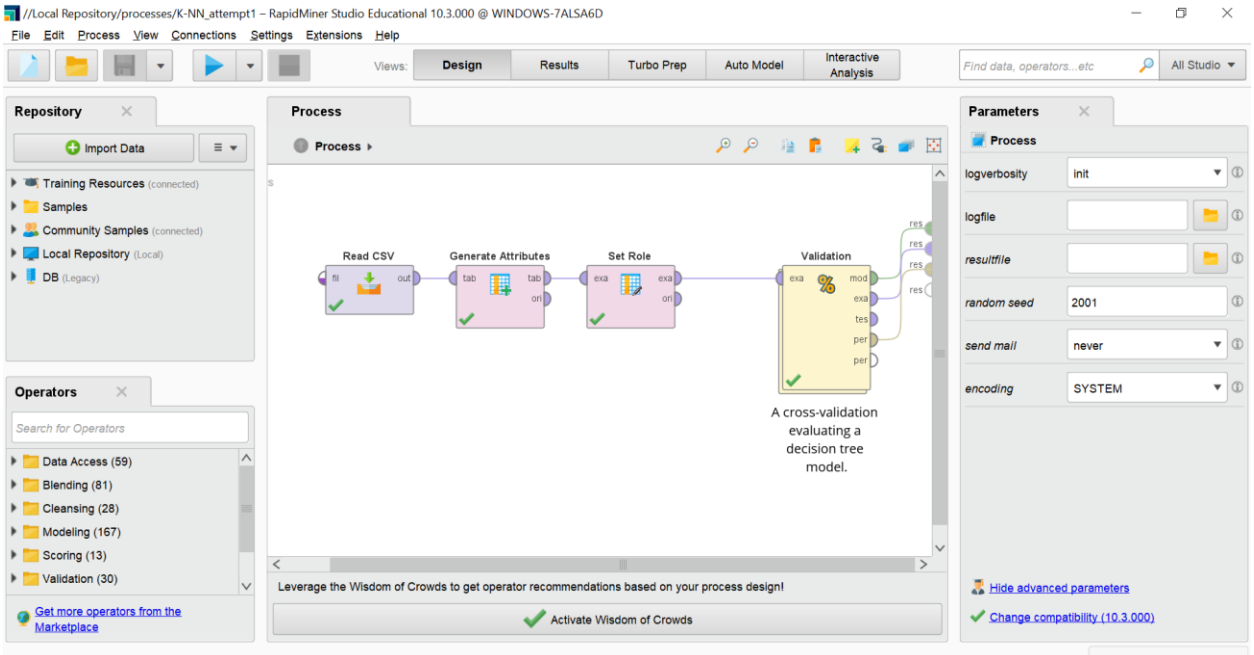
Jupyter Notebook version and files at https://github.com/lt675/IS665_Project_2

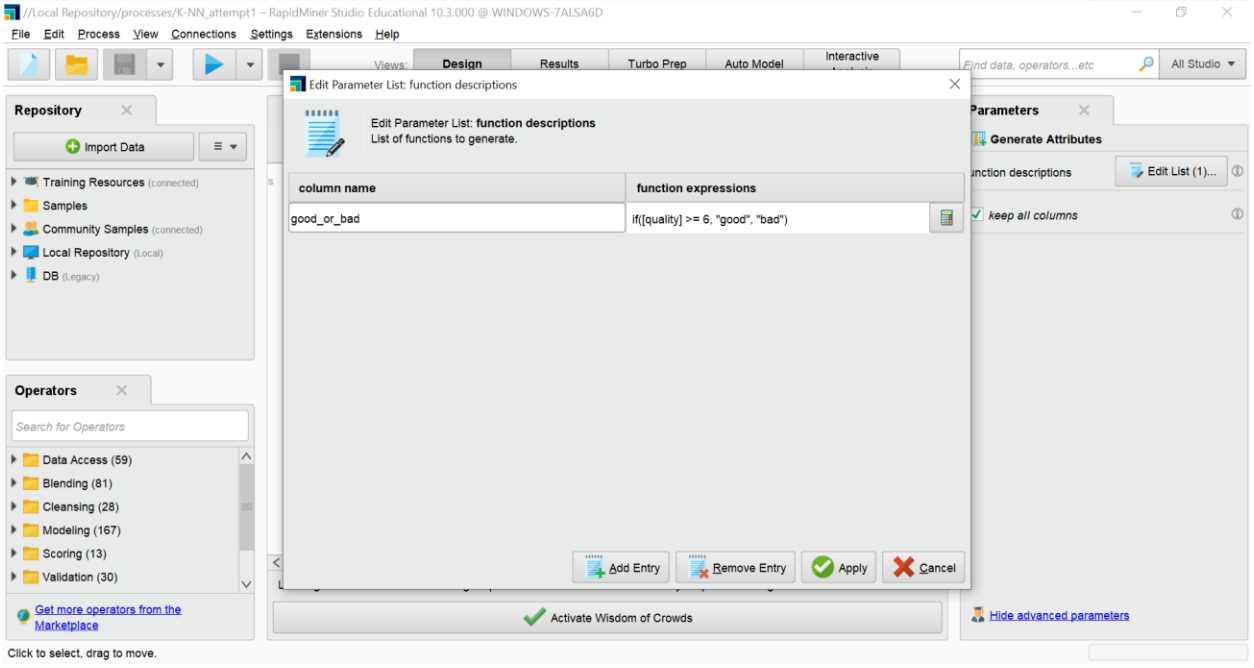
Part III. Method

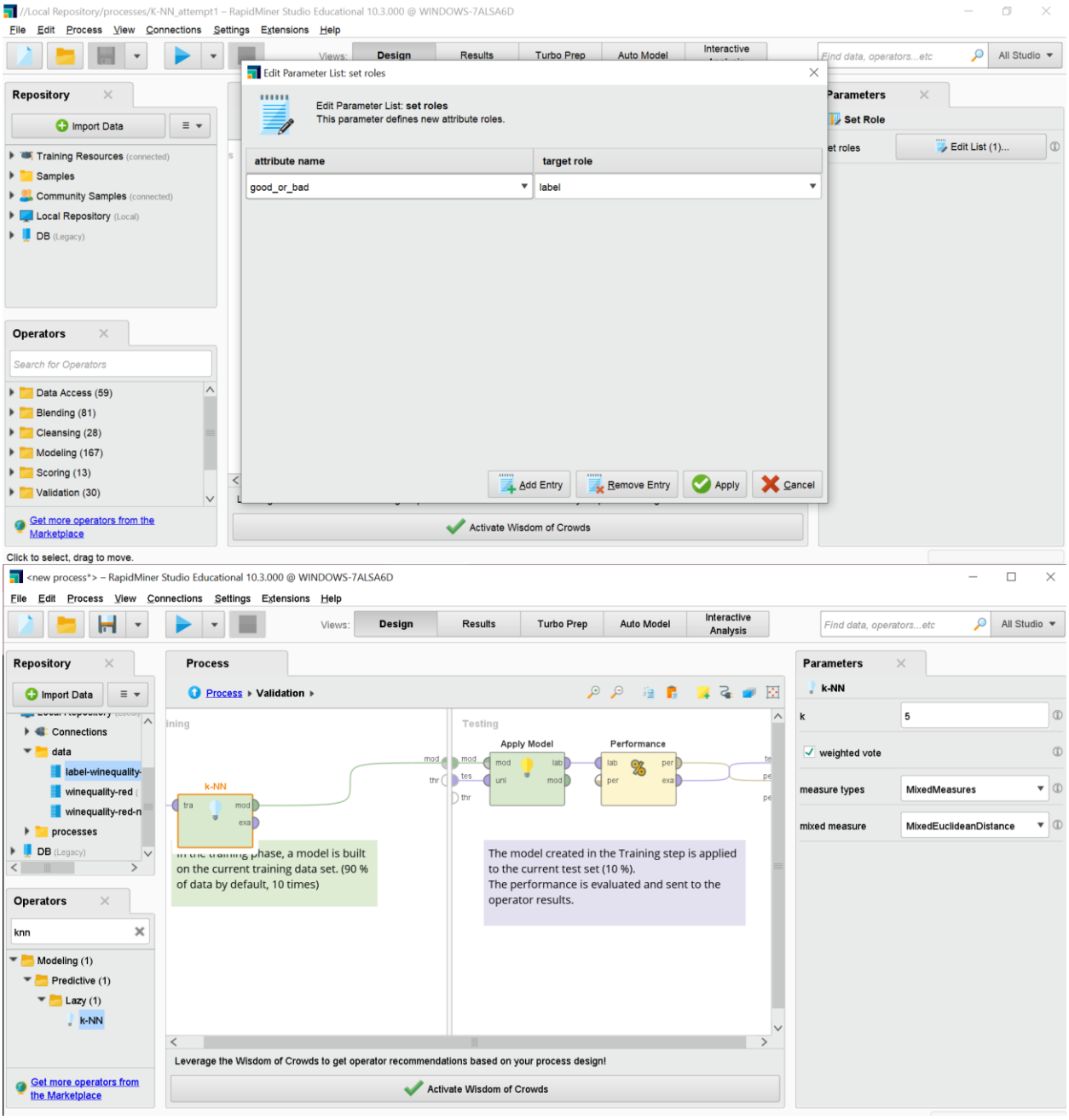
k-Nearest Neighbor (k-NN) by Winona Patrick

I applied the k-Nearest Neighbors (k-NN) algorithm, with a specific focus on k=2, to predict wine quality based on its physicochemical properties. The dataset used for this analysis contains both 'good' and 'bad' classifications of wine, determined through various physicochemical measurements.

Parameter setup for my k-NN model






Edit Parameter List: set roles
Edit Parameter List: set roles
This parameter defines new attribute roles.
attribute name
target role
good_or_bad
label
Add Entry
Remove Entry
Apply
Cancel
Repository
Import Data
Training Resources (connected)
Samples
Community Samples (connected)
Local Repository (Local)
DB (Legacy)
Operators
Search for Operators
Data Access (59)
Blending (81)
Cleansing (28)
Modeling (167)
Scoring (13)
Validation (30)
Get more operators from the Marketplace
Activate Wisdom of Crowds
Parameters
Set Role
Edit List (1)...
Click to select, drag to move.
<new process*> – RapidMiner Studio Educational 10.3.000 @ WINDOWS-7ALSA6D
Process
Process ▸ Validation ▸
Testing
Apply Model
Performance
k-NN
In the training phase, a model is built on the current training data set. (90 % of data by default, 10 times)
The model created in the Training step is applied to the current test set (10 %).
The performance is evaluated and sent to the operator results.
Leverage the Wisdom of Crowds to get operator recommendations based on your process design!
Activate Wisdom of Crowds
Connections
data
label-winequality-
winequality-red
winequality-red-n
processes
knn
Modeling (1)
Predictive (1)
Lazy (1)
k-NN
Get more operators from the Marketplace
k-NN
k
5
weighted vote
measure types
MixedMeasures
mixed measure
MixedEuclideanDistance

Fine-tuning 'k' for KNN and Rationale for 'k=2'

In the pursuit of refining the k-NN classifier for predicting wine quality, I systematically fine-tuned the 'k' parameter across various values, evaluating the model's performance via cross-validation. The obtained accuracy scores, accompanied by their corresponding standard deviation variability, demonstrated distinct performances for different 'k' values:

- 'k' of 2: accuracy: 81.68% +/- 2.65% (micro average: 81.68%)
- 'k' of 3: accuracy: 80.42% +/- 3.75% (micro average: 80.43%)
- 'k' of 4: accuracy: 80.42% +/- 3.75% (micro average: 80.43%)
- 'k' of 5: accuracy: 76.17% +/- 3.33% (micro average: 76.17%)
- 'k' of 6: accuracy: 78.68% +/- 2.06% (micro average: 78.67%)
- 'k' of 7: accuracy: 74.42% +/- 3.62% (micro average: 74.42%)

- k = 2:
    - Accuracy: 81.68% ± 2.65%
    - Precision: 80.68% (bad), 82.52% (good)
    - Recall: 79.70% (bad), 83.39% (good)
- k = 3:
    - Accuracy: 80.42% ± 3.75%
    - Precision: 77.88% (bad), 82.81% (good)
    - Recall: 80.91% (bad), 80.00% (good)
- k = 4:
    - Accuracy: 80.42% ± 3.75%
    - Precision: 78.88% (bad), 81.90% (good)
    - Recall: 79.30% (bad), 81.52% (good)
- k = 5:
    - Accuracy: 76.17% ± 3.33%
    - Precision: 73.66% (bad), 78.49% (good)
    - Recall: 75.94% (bad), 76.37% (good)
- k = 6:
    - Accuracy: 78.68% ± 2.06%
    - Precision: 76.48% (bad), 80.67% (good)
    - Recall: 78.23% (bad), 79.06% (good)
- k = 7:
    - Accuracy: 74.42% ± 3.62%
    - Precision: 71.56% (bad), 77.13% (good)
    - Recall: 74.73% (bad), 74.15% (good)

Considering these factors, k = 2 was determined to be the best choice because:

- It has the highest accuracy (81.68%), indicating the best overall performance.
- It shows a good balance between precision and recall for both classes (bad and good wines). High precision and recall for both classes are desirable, as it means the model accurately identifies both good and bad wines and does so consistently.
- The standard deviation in accuracy is relatively low (2.65%), suggesting that the performance of the model is quite consistent.

- Although the standard deviation for k = 6 is slightly lower, the overall accuracy and balance between precision and recall for k = 2 are superior.
- These results show the influence of varying 'k' values on the model's predictive accuracy. Notably, as 'k' increased, a discernible decline in accuracy emerged, indicating a delicate balance between model complexity and its ability to generalize.
- The decision to favor 'k=2' is reinforced by its consistent and competitive accuracy, substantiated across both the initial dataset and an independent external dataset assessment.

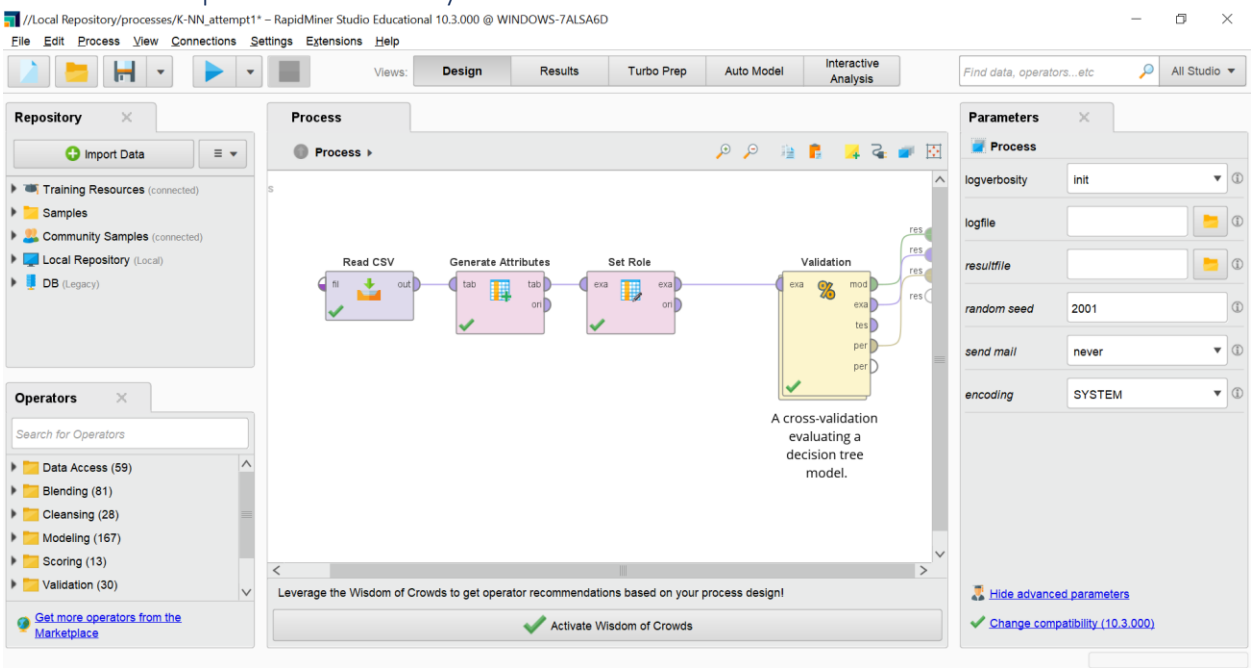
Cross-Dataset Performance Consistency

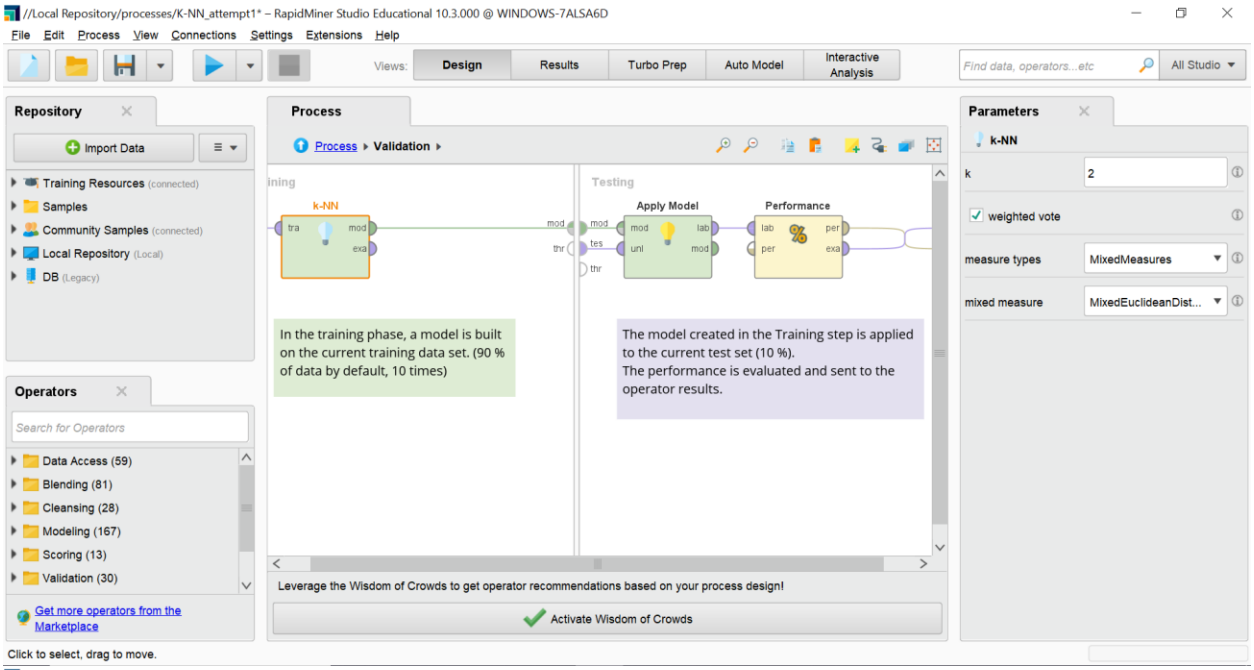
Moreover, I subjected the 'k=2'-optimized model to a separate white wine dataset evaluation (accessible via the provided link alongside the red wine dataset). This external test yielded a very similar accuracy of accuracy: 81.40% +/- 1.32% (micro average: 81.40%). The robust alignment in performance metrics between the initial dataset and this external validation test underscores the model's reliability and consistency when 'k=2'.

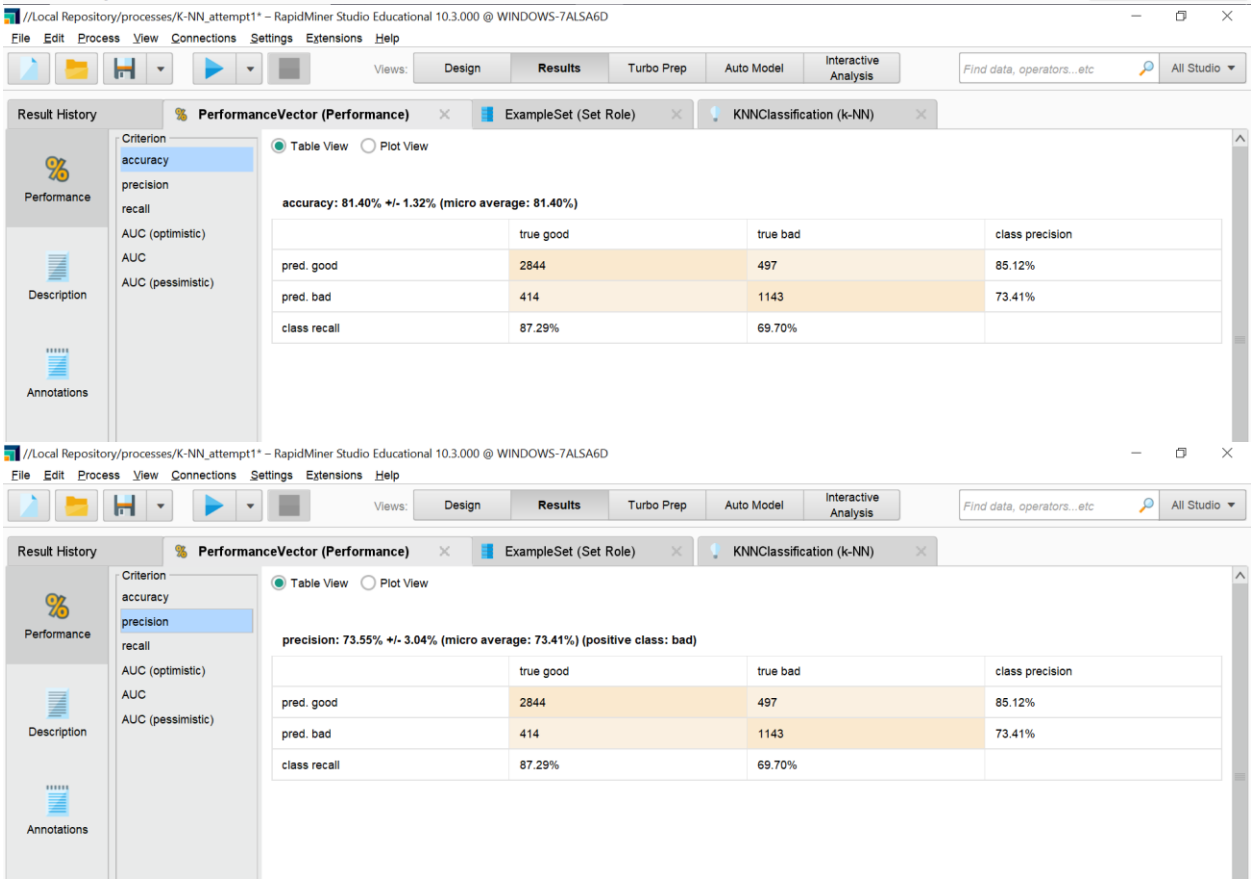
Stability and Generalizability

The stability showcased across diverse datasets and the strong performance of 'k=2' without compromising generalizability firmly justify its selection. This choice resonates with our objective: to create a model adept at capturing intricate patterns within the training data while demonstrating robustness when faced with previously unseen instances. The coherent alignment in accuracy between datasets and the commendable performance stability solidifies the rationale behind favoring 'k=2' as the optimal parameter for my k-NN model.

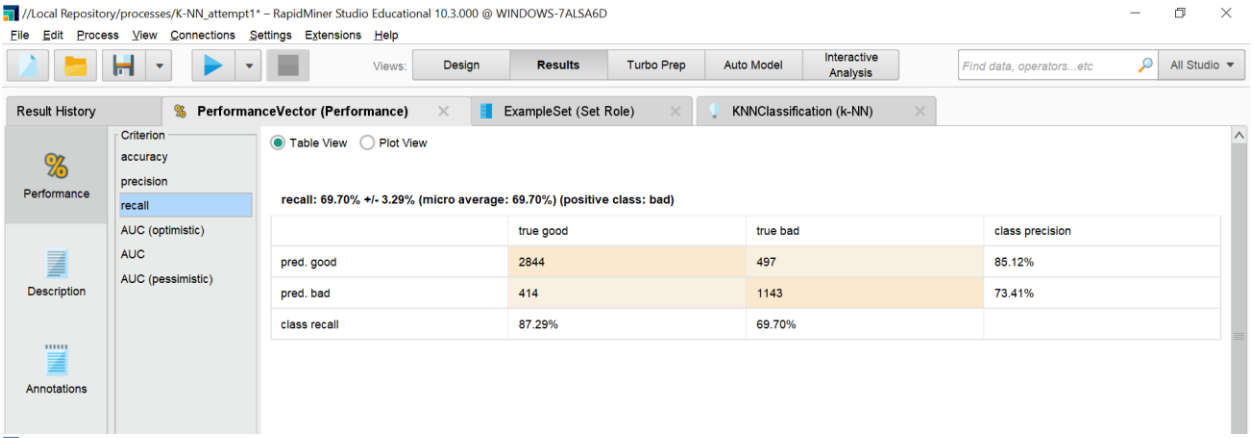
Parameter setup and results for my test white wine test dataset

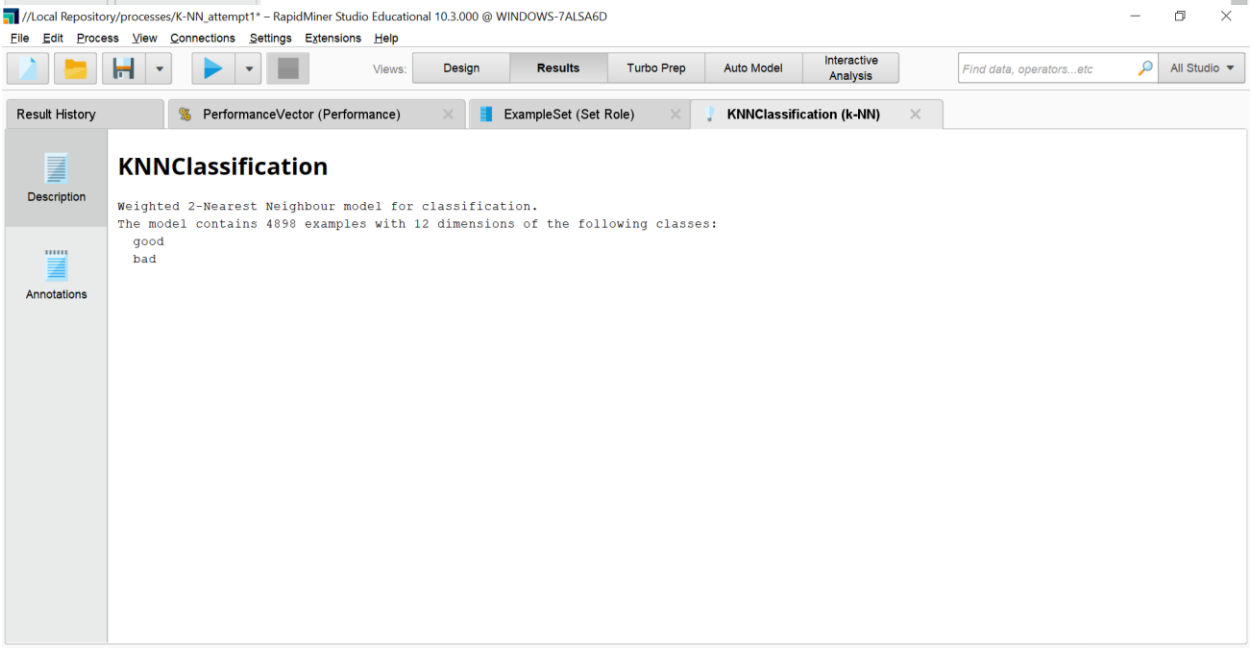



//Local Repository/processes/K-NN_attempt1* – RapidMiner Studio Educational 10.3.000 @ WINDOWS-7ALSA6D
File Edit Process View Connections Settings Extensions Help
Views: Design Results Turbo Prep Auto Model Interactive Analysis
Find data, operators...etc All Studio
Repository
Import Data
Training Resources (connected)
Samples
Community Samples (connected)
Local Repository (Local)
DB (Legacy)
Process
Process ▸ Validation ▸
k-NN
Testing
Apply Model
Performance
In the training phase, a model is built on the current training data set. (90 % of data by default, 10 times)
The model created in the Training step is applied to the current test set (10 %).
The performance is evaluated and sent to the operator results.
Parameters
k-NN
k 2
weighted vote
measure types MixedMeasures
mixed measure MixedEuclideanDist...
Operators
Search for Operators
Data Access (59)
Blending (81)
Cleansing (28)
Modeling (167)
Scoring (13)
Validation (30)
Get more operators from the Marketplace
Leverage the Wisdom of Crowds to get operator recommendations based on your process design!
Activate Wisdom of Crowds
Click to select, drag to move.


//Local Repository/processes/K-NN_attempt1* – RapidMiner Studio Educational 10.3.000 @ WINDOWS-7ALSA6D
File Edit Process View Connections Settings Extensions Help
Views: Design Results Turbo Prep Auto Model Interactive Analysis
Find data, operators...etc All Studio
Result History
PerformanceVector (Performance)
ExampleSet (Set Role)
KNNClassification (k-NN)
Performance
Description
Annotations
Criterion
accuracy
precision
recall
AUC (optimistic)
AUC
AUC (pessimistic)
Table View Plot View
accuracy: 81.40% +/- 1.32% (micro average: 81.40%)
true good true bad class precision
pred. good 2844 497 85.12%
pred. bad 414 1143 73.41%
class recall 87.29% 69.70%

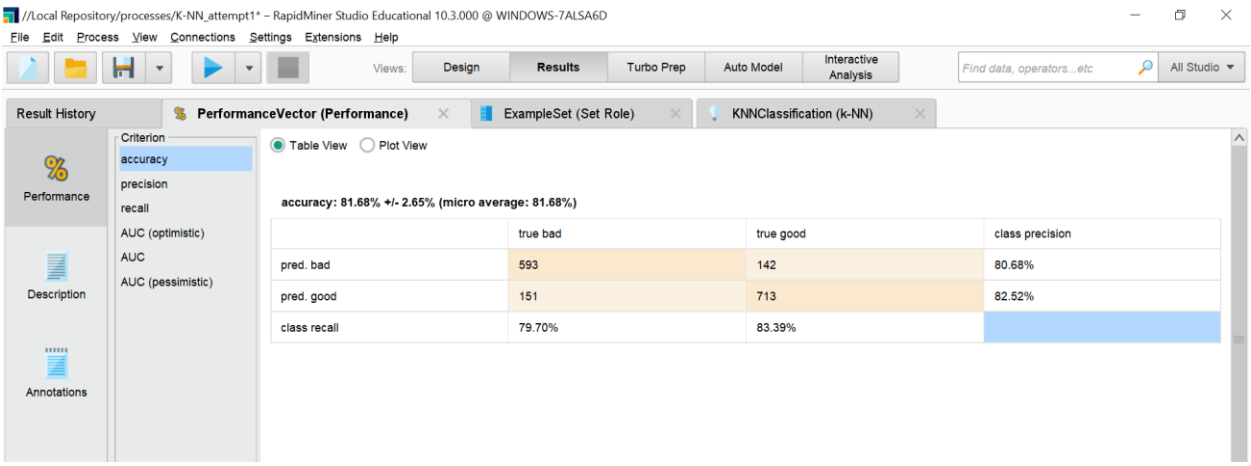
//Local Repository/processes/K-NN_attempt1* – RapidMiner Studio Educational 10.3.000 @ WINDOWS-7ALSA6D
File Edit Process View Connections Settings Extensions Help
Views: Design Results Turbo Prep Auto Model Interactive Analysis
Find data, operators...etc All Studio
Result History
PerformanceVector (Performance)
ExampleSet (Set Role)
KNNClassification (k-NN)
Performance
Description
Annotations
Criterion
accuracy
precision
recall
AUC (optimistic)
AUC
AUC (pessimistic)
Table View Plot View
precision: 73.55% +/- 3.04% (micro average: 73.41%) (positive class: bad)
true good true bad class precision
pred. good 2844 497 85.12%
pred. bad 414 1143 73.41%
class recall 87.29% 69.70%


//Local Repository/processes/K-NN_attempt1* – RapidMiner Studio Educational 10.3.000 @ WINDOWS-7ALSA6D
File Edit Process View Connections Settings Extensions Help
Views: Design Results Turbo Prep Auto Model Interactive Analysis
Find data, operators...etc
All Studio
Result History
PerformanceVector (Performance)
ExampleSet (Set Role)
KNNClassification (k-NN)
Performance
Description
Annotations
Criterion
accuracy
precision
recall
AUC (optimistic)
AUC
AUC (pessimistic)
Table View
Plot View
recall: 69.70% +/- 3.29% (micro average: 69.70%) (positive class: bad)
true good
true bad
class precision
pred. good
2844
497
85.12%
pred. bad
414
1143
73.41%
class recall
87.29%
69.70%

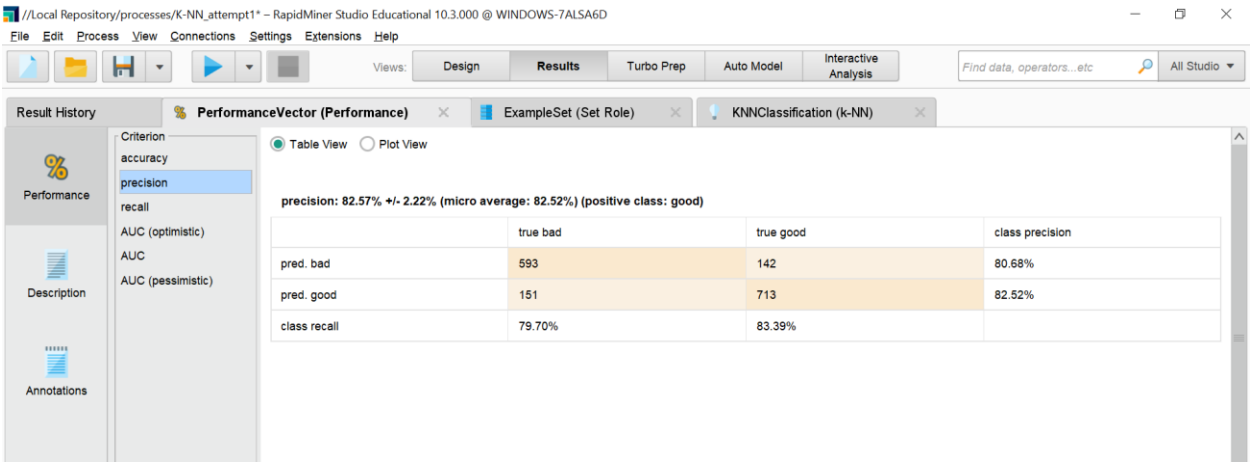

//Local Repository/processes/K-NN_attempt1* – RapidMiner Studio Educational 10.3.000 @ WINDOWS-7ALSA6D
File Edit Process View Connections Settings Extensions Help
Views: Design Results Turbo Prep Auto Model Interactive Analysis
Find data, operators...etc
All Studio
Result History
PerformanceVector (Performance)
ExampleSet (Set Role)
KNNClassification (k-NN)
Description
Annotations
KNNClassification
Weighted 2-Nearest Neighbour model for classification.
The model contains 4898 examples with 12 dimensions of the following classes:
good
bad

Results of my actual k-NN Model

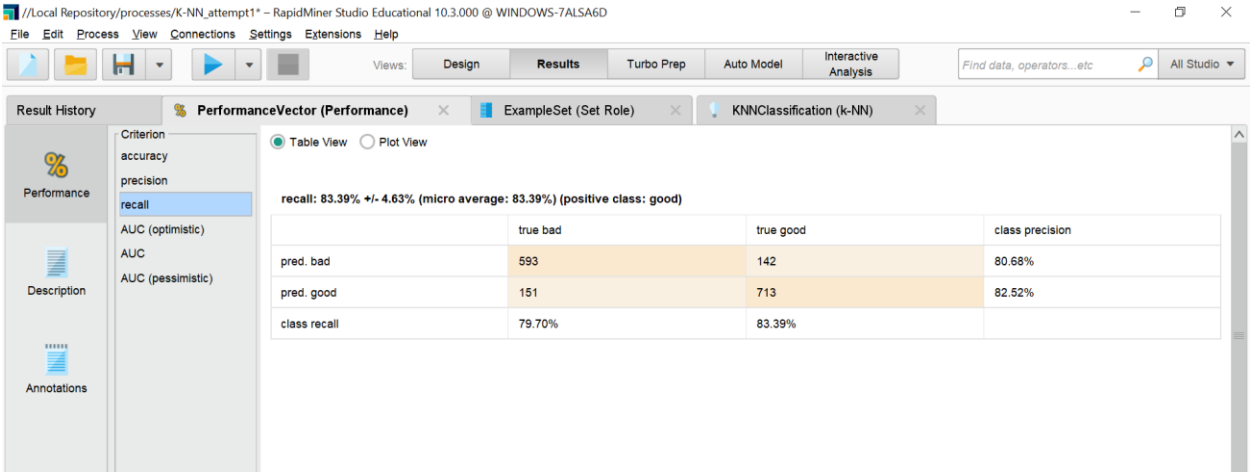
Accuracy Confusion Matrix:



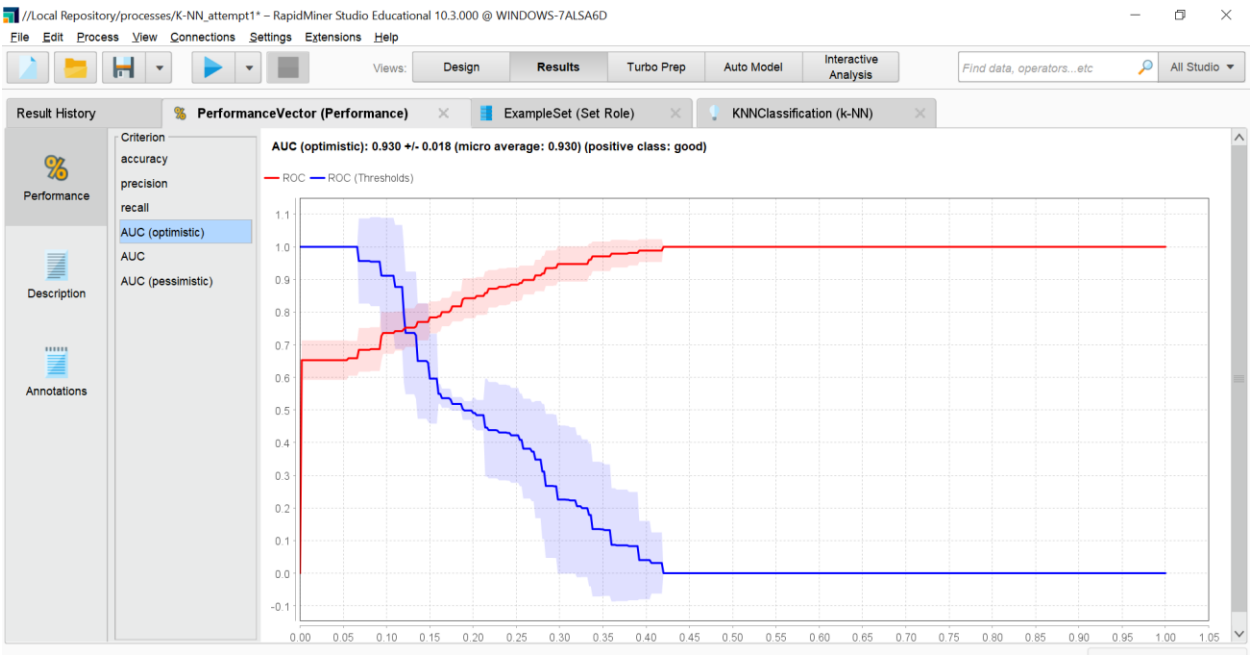
Precision Confusion Matrix:



Recall Confusion Matrix:



AUC (Optimistic)



Model Performance Overview and Interpretation

1. Overall Accuracy
    a. The model achieved an accuracy of 81.68%, with a standard deviation of 2.65%. This high accuracy rate indicates that the model correctly predicts the quality of the wine 81.68% of the time, demonstrating robust performance.
2. Precision
    a. The overall precision of the model is 82.57% (micro average: 82.52%), with a standard deviation of 2.22%. This precision rate, particularly for the positive class ('good' wine), is significant. It means that when the model predicts a wine as 'good', it is correct 82.52% of the time. The high precision reflects the model's effectiveness in minimizing false positives.
    b. Specifically, the class precision for predicting 'bad' wine is 80.68%, and for 'good' wine, it is 82.52%. These figures suggest a balanced capability of the model in identifying both classes accurately.
3. Recall
    a. The recall for the model is 83.39% (micro average: 83.39%), with a standard deviation of 4.63%. This metric is particularly noteworthy for the positive class ('good' wine). It indicates that the model is capable of identifying 83.39% of all actual 'good' wines. High recall is important as it implies fewer false negatives, meaning the model rarely misses identifying good quality wines.
    b. The class recall rates are 79.70% for 'bad' wine and 83.39% for 'good' wine, indicating a slightly better performance in correctly identifying all 'good' wines compared to 'bad' wines.

4. AUC (optimistic) graph:
    a. ROC Curve: The blue line represents the ROC curve of my model for a particular class labeled as 'good'.
    b. AUC: My model has an AUC (optimistic) of 0.930 ± 0.018, as indicated in the description box. This means my model has a high ability to distinguish between the positive class ('good') and the negative class. The AUC is a single scalar value that summarizes the overall performance of the classifier; the closer the value is to 1, the better it is.
    c. Optimistic and Pessimistic AUC: The micro average AUC is very high at 0.930, indicating that the model performs well in distinguishing between the classes across all thresholds.
    d. Overall, this graph indicates that my model has a high level of performance in classifying the quality of wine as 'good', with a ROC curve significantly above the diagonal line of no-discrimination (which would represent a completely random classifier). The AUC value being close to 1 reinforces the model's strong discriminative power.

The chosen k-NN model with k=2 demonstrates strong performance in predicting wine quality. The high accuracy suggests overall effectiveness, while the balance between precision and recall indicates the model's capability in correctly identifying and minimizing false classifications for both 'good' and 'bad' wines.

The model's precision suggests a strong predictive power, especially in correctly identifying good quality wines, which is crucial for applications where the cost of falsely identifying a bad wine as good is high. Similarly, the high recall indicates the model's strength in capturing most of the good quality wines, essential in scenarios where missing a good quality wine is undesirable.

The slight variation in performance between the 'good' and 'bad' classifications (as seen in the precision and recall values) may be attributed to the inherent characteristics of the dataset or the distribution of the classes.

A Case Study in Data Mining By: Simranjit Singh

This report presents a comprehensive analysis of a wine quality dataset, aimed at predicting the quality of wine based on various physicochemical properties. The dataset is sourced from the UCI Machine Learning Repository and provides an intriguing exploration into the world of wine analytics.

The dataset comprises observations of red and white wine samples, each characterized by specific attributes such as fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free and total sulfur dioxide, density, pH, sulfates, alcohol content, and overall quality. These features offer a multifaceted view of wine composition, allowing for a detailed examination of how these variables interact to define the quality of wine.

Fixed Acidity	Volatile Acidity	Citric Acid	Residual Sugar	Chlorides	Free Sulfur Dioxide	Total Sulfur Dioxide	Density	pH	Sulphates	Alcohol	Quality	Color
7.4	0.7	0	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5	Red
7.8	0.88	0	2.6	0.098	25	67	0.9968	3.2	0.68	9.8	5	Red
7.8	0.76	0.04	2.3	0.092	15	54	0.997	3.26	0.65	9.8	5	Red
11.2	0.28	0.56	1.9	0.075	17	60	0.998	3.16	0.58	9.8	6	Red
7.4	0.7	0	1.9	0.076	11	34	0.9978	3.51	0.56	9.4	5	Red

The primary objective of this report is to apply data mining algorithms to this dataset to predict wine quality effectively. By employing methods such as logistic regression, k-Nearest Neighbors (kNN), and Support Vector Machines (SVM), this study aims to uncover the underlying patterns in the data that influence wine quality. The results of this analysis will provide insights into the critical factors contributing to the quality of wine, thereby aiding vintners and consumers alike in understanding and predicting wine characteristics based on its chemical composition.

Data Preparation and Preprocessing

Data Acquisition: The dataset was obtained from the UCI Machine Learning Repository using the URL 'https://archive.ics.uci.edu/static/public/186/data.csv'. Python's requests library was employed to fetch the data, which was then read into a pandas DataFrame for ease of manipulation. The initial code snippet for this process is as follows:

```
# URL of the CSV data
```

```
csv_url = 'https://archive.ics.uci.edu/static/public/186/data.csv'

# Fetch data from the URL
response = requests.get(csv_url)
data = StringIO(response.text)

# Read the CSV data
df = pd.read_csv(data)
```

Preprocessing Steps

The dataset underwent several preprocessing steps to ensure it was suitable for analysis and modeling:

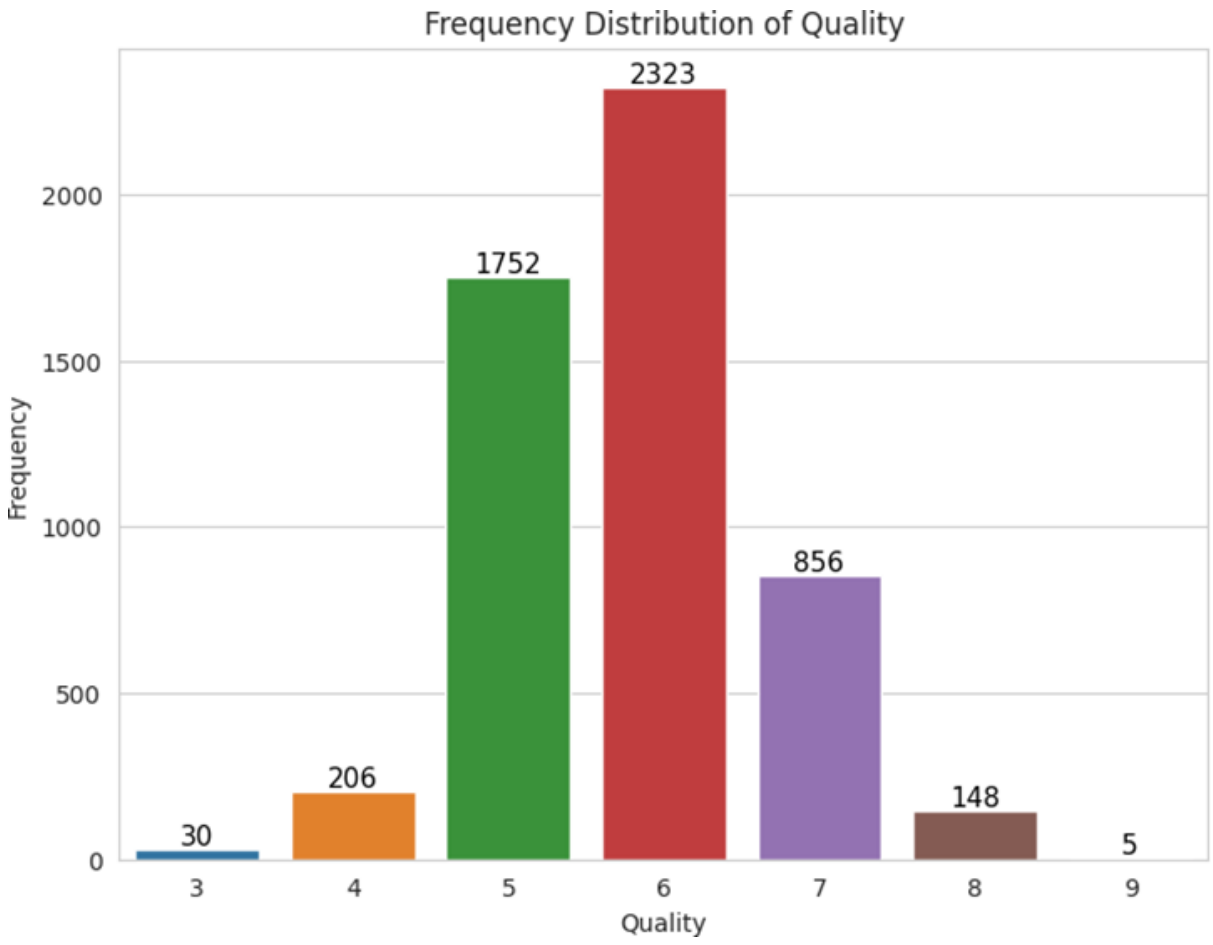
- Renaming Columns: To enhance readability and ease of access, column names were abbreviated or modified. For instance, 'fixed_acidity' was renamed to 'FA', 'volatile_acidity' to 'VA', and so on. This renaming was performed using pandas' rename function:
- Converting 'color' to Binary: The 'color' column, indicating the type of wine, was transformed into a binary format for analytical convenience: 'red' was coded as 0, and 'white' as 1. This binary encoding is crucial for modeling as it converts categorical data into a numeric format that can be processed by machine learning algorithms.
- Dropping Duplicates: To ensure the uniqueness of each data entry and remove potential biases caused by duplicated records, the dataset was scanned for and purged of duplicates.

```
df = df.rename(columns={
    'fixed_acidity': 'FA',
    'volatile_acidity': 'VA',
    # ... other columns ...
    'color': 'Col'
})

df['Col'] = df['Col'].map({'red': 0, 'white': 1})

df = df.drop_duplicates()
```

Exploratory Data Analysis (EDA)

Quality Distribution Analysis

A critical part of the EDA was analyzing the distribution of wine quality in the dataset. A count plot was generated using seaborn's countplot function to visualize the frequency distribution of different quality ratings. This plot provides insights into the balance or imbalance of the dataset concerning the quality of wine.
The resulting plot (as shown in the screenshot) reveals the distribution of wine qualities in the dataset. Notably, this visualization helps in understanding the skewness or balance in the quality ratings, which is crucial for the subsequent modeling and analysis.



Feature Engineering and Data Splitting

Binarization of the 'Quality' Variable

In the dataset, the 'quality' variable represents the quality rating of the wine on a scale. For the purposes of this analysis, it was transformed into a binary variable to facilitate a binary classification task. This binarization process involved categorizing wines as 'high quality' (1) or 'not high quality' (0), based on a threshold. The median value of the 'quality' ratings was used as the threshold to ensure a balanced binary classification:

Feature and Target Variable Selection

For the predictive modeling, all columns except the original 'quality' and the newly created 'quality_binary' were used as features (X). The 'quality_binary' column served as the target variable (y). This separation of features and target variable is essential for supervised learning tasks:

Data Splitting and Standardization

The dataset was split into training and testing sets to evaluate the performance of the models on unseen data. A common split ratio of 70% training and 30% testing was used. Additionally, feature standardization was performed to normalize the data, enhancing the performance and stability of the models, particularly for algorithms sensitive to the scale of input features like logistic regression and kNN:

```
# Binarize the quality variable (By taking Median value)
df['quality_binary'] = df['Qual'].apply(lambda x: 1 if x >= 6 else 0)

# Define features and target variable for classification
X = df.drop(['Qual', 'quality_binary'], axis=1)  # Drop non-feature columns
y = df['quality_binary']

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=0)

# Standardize the features (necessary for logistic regression and kNN)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Model Selection and Training

Introduction to Algorithms

Three different machine learning algorithms were chosen for this study:

- Logistic Regression: A fundamental classification algorithm suitable for binary classification tasks.
- k-Nearest Neighbors (kNN): This non-parametric method was chosen for its simplicity and effectiveness in classification tasks.
- Support Vector Machine (SVM): Known for its robustness and efficacy in high-dimensional spaces, making it suitable for datasets with multiple features.

These models were selected to provide a comprehensive analysis across different types of algorithms, offering insights into their varying performances on the same dataset.

Model Initialization and Training

Each model was initialized and then trained on the scaled training dataset. The following code snippets illustrate these steps:

```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression()
log_reg.fit(X_train_scaled, y_train)

from sklearn.neighbors import KNeighborsClassifier

knn_euclidean = KNeighborsClassifier(n_neighbors=5)
knn_euclidean.fit(X_train_scaled, y_train)

from sklearn.svm import SVC

svm_model = SVC()
svm_model.fit(X_train_scaled, y_train)
```

Each model was fitted using the scaled training data. This fitting process involves adjusting the model parameters to minimize the error between the predicted and actual values on the training data.

Model Evaluation and Results

Classification Reports

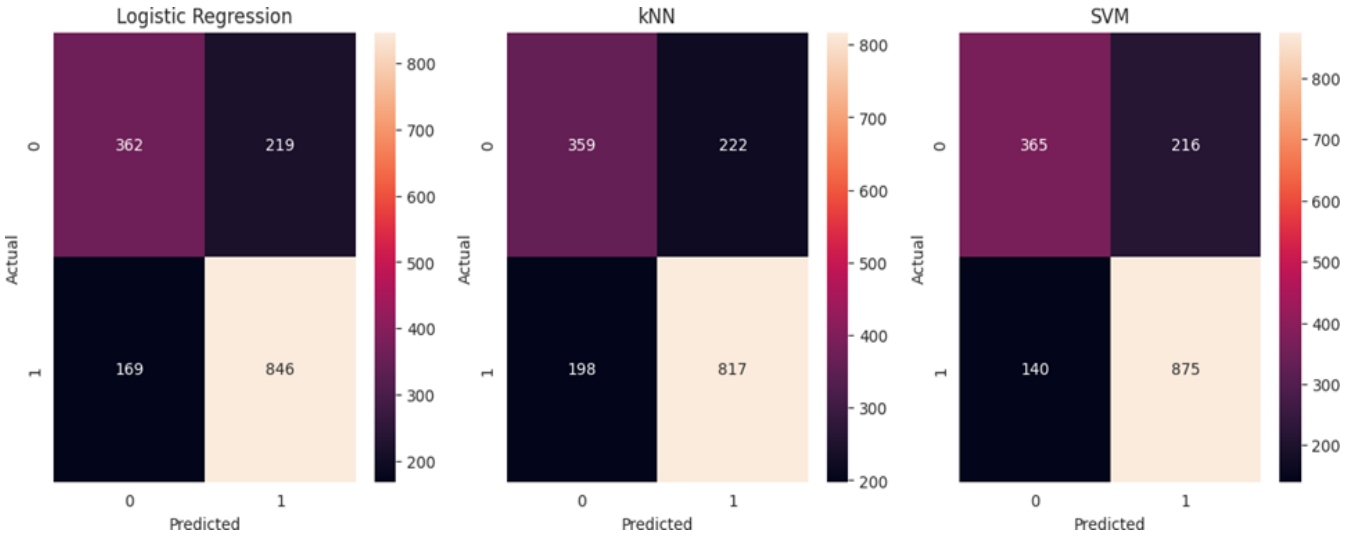
The performance of each model was evaluated using classification reports and confusion matrices. The classification reports provide key metrics like precision, recall, f1-score, and accuracy for each class, which are crucial for understanding model performance, especially in imbalanced datasets.

```
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.68      0.62      0.65       581
           1       0.79      0.83      0.81      1015

    accuracy                           0.76      1596
   macro avg       0.74      0.73      0.73      1596
weighted avg       0.75      0.76      0.75      1596

kNN Classification Report:
              precision    recall  f1-score   support
```

```
           0       0.64      0.62      0.63       581
           1       0.79      0.80      0.80      1015

    accuracy                           0.74      1596
   macro avg       0.72      0.71      0.71      1596
weighted avg       0.73      0.74      0.74      1596

SVM Classification Report:
              precision    recall  f1-score   support

           0       0.72      0.63      0.67       581
           1       0.80      0.86      0.83      1015

    accuracy                           0.78      1596
   macro avg       0.76      0.75      0.75      1596
weighted avg       0.77      0.78      0.77      1596
```

Confusion Matrices

Confusion matrices for each model provide a visual and numerical representation of the model performance, especially highlighting the cases of false positives and false negatives.



Comparative Analysis

- Accuracy: SVM showed the highest overall accuracy, indicating its effectiveness in correctly classifying the majority of instances.
- Precision and Recall: Logistic Regression and SVM demonstrated high precision, while kNN was slightly lower. In terms of recall, SVM again outperformed the other two models.
- F1-Score: The F1-score, which balances precision and recall, was highest for SVM, followed by Logistic Regression and kNN.

Reflection on Model Strengths and Weaknesses

- Logistic Regression showed balanced precision and recall but was slightly less effective in handling the dataset's complexity.
- kNN struggled with precision, which could be due to the model's sensitivity to the feature space and scaling.
- SVM outperformed in most metrics, likely due to its ability to handle high-dimensional data and find optimal separation between classes.

Conclusion and Recommendations

Key Findings

- Among the evaluated models, SVM demonstrated the best performance across multiple metrics, including accuracy, precision, recall, and F1-score.
- The standardization of features played a significant role in the performance of the models, especially for kNN and Logistic Regression.
- The binarization of the quality variable facilitated a straightforward binary classification approach, allowing for clear evaluation metrics.

Best Performing Model

- SVM emerged as the most effective model for this dataset. Its ability to manage the complexity and high dimensionality of the data contributed significantly to its superior performance.

Recommendations for Future Work

- Parameter Tuning: Experimenting with hyperparameter tuning for each model, particularly kNN (number of neighbors) and SVM (kernel types and C value), could further enhance performance.
- Feature Engineering: Investigating more sophisticated feature engineering techniques, like interaction terms or polynomial features, might reveal more insights.
- Comparing with Other Models: Extending the analysis to include other models, such as decision trees or ensemble methods like Random Forests, could provide a broader understanding of the dataset.
- Addressing Imbalance: If any class imbalance is detected in the dataset, techniques such as SMOTE or weighted class balancing in models could be explored.

Conclusion

In conclusion, the k-NN model with k=2 proves to be highly effective in predicting the quality of wine based on physicochemical properties. Its high accuracy, along with balanced precision and recall, makes it a reliable tool in the classification of wine quality. This model can be instrumental for winemakers, quality control labs, and the wine industry in general, to gauge wine quality efficiently and accurately. However, I believe it important to note that the k-NN algorithm, while effective, does not provide insights into which physicochemical properties are most influential in determining wine quality. For industries or researchers interested in understanding the underlying factors affecting wine quality, other models with better interpretability might be more appropriate. However, for the purposes of our research question, the model is fitting and can distinguish between “good” and “bad” wine qualities well.

Random Forest by Li Temeran

Setup and Discussion

I chose the random forest algorithm based on it being an aggregation of decisions trees, which seemed like an interesting topic to explore. It introduces the concept of bagging or bootstrap aggregation, where sets of data are created with replacement, so individual data points are present in different sets. Random forest extends this further by also assigning random features to each set. This creates uncorrelated decision trees that can vote based on their particular dataset and the most common output will be the one with the most votes. It operates under the concept of the wisdom of the crowd. This should also make it rigorous to outliers, which I decided to test.

Initial Classification

I started with an initial classification using most of the base parameters associated with the Random Forest Classifier. The training was spilt set as 70% of the data, and the random_state was set at 42. Random_state ensures that each set will contain the same data each time, so it can be reproduced. If set at 'None', it will create different datasets or bags each time. The integer values determine how many bags there will be. The most common value used is 42, due to it being the answer the answer given by a supercomputer to "the Ultimate Question of Life, the Universe, and Everything" in Douglas Adams' "The Hitchhiker's Guide to the Galaxy," 5-part trilogy. A friend called it 'engineer humor'.

The classifier was run at n_jobs=-1 so that all processing power can be used, and included the oob_score, or out of bag score. This gives an initial accuracy score of the data points that will be used in the test set.

Hyperparameter Tuning

Hyperparameter tuning was then performed using GridSearchCV. It went through every possible parameter I decided to test and determined the best parameter set to use.

I decided to adjust the parameters below:

- max_depth: maximum depth of each individual decision tree; default = None
- min_samples_leaf: minimum number of samples in each leaf of the tree; default = 1
- n_estimators: number of trees to consider; default = 100
- max_features: maximum number of features provided to each tree; default = "sqrt"
- min_samples_split: minimum number of samples needed to split a leaf; default = 2

I left criterion as gini since I don't believe it increases accuracy substantially, and it requires less processing power. Now that I know how long the gridsearch takes, I'm glad I left it at gini. I left bootstrap at its default of true since I wanted the train/test split. I decided to use the oob_score since I thought it may help with the accuracy of the algorithm. In the future, I may test the difference between whether it makes a difference.

I think the most interesting of the parameters that I decided to check was the max_features. I likely didn't need to add "log2" since it is fairly close to the square root. It's something I'll consider more in the

future. Since the values for those are around 3, I decided to use 5, 7, and 9 to see if there was an increase in accuracy.

```
# Hyperparameter tuning
rf = RandomForestClassifier(random_state=42, n_jobs=-1, oob_score=True)

params = {
    'max_depth': [2, 3, 5, 10, 20, 40, 50],
    'min_samples_leaf': [1, 2, 5, 10, 20, 50],
    'n_estimators': [10, 25, 30, 50, 100, 200, 350, 500, 750, 1000],
    'max_features': [5, 7, 9, 'sqrt', 'log2'],
    'min_samples_split': [2, 4, 5, 10]
}

# Instantiate the grid search model
grid_search = GridSearchCV(estimator=rf,
                           param_grid=params,
                           cv=4,
                           n_jobs=-1, verbose=3, scoring="accuracy")

'''Fitting 4 folds for each of 8400 candidates, totaling 33600 fits
This took a long time to compile. In the future, prune the parameters,
 lower the cv, or get a better computer'''
```

As I mentioned in my documentation, I need to use less choices or get a computer with more processing power. It determined the best parameters to be the base ones with these changes. Since it took so much computational power to run this, I used these parameters for all versions I ran.

```
RandomForestClassifier(max_depth=40, n_estimators=30, n_jobs=-1,
oob_score=True,
                       random_state=42)
```

Random Forest Multiclassification

This was the most straightforward to run. I only had to define the independent and dependent variable (label). I used the same train/test split from parameter selection.

```
# Define independent variables by removing quality
X = df.drop('quality', axis=1)

# Define dependent variable or label
y = df['quality']
```

Random Forest Multiclassification with Outlier Removal

This brought in a statistical element of the z-score, which standardizes the data and determines its distance from the mean. First, I determined z-scores of the data points and changed them to their absolute value. I chose to remove data points with a z-score greater than 3. This decision is based on the 68-95-99.7 rule where 99.7% of the data points within a set fall within 3 standard deviations of the mean. Again, I used the same train/test split from the parameter section

```
# Find z-scores to determine outliers. IQR method is too time-consuming.
z = np.abs(stats.zscore(df))

# Remove data points with a z-score greater than 3
# Based on the 68-95-99.7 rule, this should leave us with points that lie
within
# 99.7% of the data since those are within 3 standard deviations of the mean.
df_o = df[(z < 3).all(axis=1)]

# Define independent variables by removing quality
X = df_o.drop(columns='quality')

# Define dependent variable or label
y = df_o['quality']
```

Random Forest Binary Classification

Since some of my teammates were using binary classification out of choice or necessity, I ran another classifier with a binary classifier. The data was split somewhat evenly by classifying wines with a score of below 6 (<6) or 6 and above (>=6). Once again, I used the same train/test split from the parameter section.

```
# Define binary data split and create new dataframe
ddf = pd.DataFrame()
ddf['class'] = df['quality']
ddf.loc[(ddf['class'] < 6), 'new_class'] = 0
ddf.loc[(ddf['class'] >= 6), 'new_class'] = 1

# Concatenate original and data split dataframes
df_bin = pd.concat((ddf, df), axis=1)

# Remove unnecessary columns
df_bin = df_bin.drop(['class'], axis=1)
df_bin = df_bin.drop(['quality'], axis=1)

# Define independent variables by removing new class column
X = df_bin.drop('new_class', axis=1)

# Define dependent variable or label
y = df_bin['new_class']
```

Algorithm Conclusion

Distribution

The algorithm picked a good distribution of samples as far as proportions of the total samples with regards to quality. I feel that this makes the data prediction more reliable. It is interesting to note that there are no data points with a quality score of 3 in the multiclass version with outliers removed. This may be a product of there being so few wines with a quality score of 3, or it may be that they were all considered outliers by the z-score removal.

Distribution of Test Data Points

	Original		Multiclass		Multi - Outliers	
Quality Score	Count	Percent of Total	Count	Percent of Total	Count	Percent of Total
3	10	0.63%	1	0.21%	0	0.00%
4	53	3.31%	17	3.54%	15	3.44%
5	681	42.59%	195	40.63%	179	41.06%
6	638	39.90%	200	41.67%	179	41.06%
7	199	12.45%	61	12.71%	58	13.30%
8	18	1.13%	6	1.25%	5	1.15%
Total	1599		480		436	

	Original		Binary Class	
Quality Score	Count	Percent of Total	Count	Percent of Total
< 6	744	46.53%	213	44.38%
>= 6	855	53.47%	267	55.63%
Total	1599		480	

Classification Reports

The multiclass versions had very similar outcomes across precision, recall, F1-score, and accuracy so it would stand to reason that outlier removal doesn't matter. This supports the notion that random forest is robust to outliers since it is an aggregation of decisions trees. The accuracy for multiclass is around 65% between both versions, which isn't bad for a question where there isn't a huge amount at stake in the grand scheme of life. I would not consider this good for something like diagnosis of disease. The algorithm is good at predicting 5, 6 and 7 scoring wines. The recall is much higher in 5 and 6 scoring wines, so it was better at identifying positives among those that were actually positive than those that it classified as positive. The opposite is true for 7 scoring wines. The f1-scores speak for themselves though in that it decreases from 5 through 7. It is interesting to note that despite there being similar counts of 5 and 6 scoring wines, there is such a disparity in metrics. There may be some underlying feature that could be investigated further, which may aid in further tuning the algorithm. Since there are so few 3, 4 and 8 scoring wines, it is difficult to evaluate the performance of the algorithm for that subset.

Due to the imbalanced distribution of the quality scores, I'm not putting much stock in the macro average of the classification metrics. This is likely why they are so low. The weighted average metrics reflect the data more clearly. Since the general recall is higher,

The binary version of the algorithm saw a significant increase in all metrics. The accuracy is almost 80%. This makes sense along the lines of increasing the odds of getting something right by decreasing the choices. The algorithm is certainly better at predicting wines with a score greater than 6. Again, there are differences between the precision and recall rates of the two choices which also lends itself to further investigation of the features of the wines themselves. Whether to use the multi or binary class is really up to the user and the business question at hand.

Classification Reports

Multiclass Random Forest Algorithm

	Precision	Recall	F1-Score	Support
3	0.00%	0.00%	0.00%	1
4	0.00%	0.00%	0.00%	17
5	72.77%	75.38%	74.06%	195
6	63.01%	69.00%	65.87%	200
7	57.89%	54.10%	55.93%	61
8	50.00%	16.67%	25.00%	6
Accuracy	66.46%	66.46%	66.46%	66.46%
Macro Avg	40.61%	35.86%	36.81%	480
Weighted Avg	63.80%	66.46%	64.95%	480

Multiclass Random Forest Algorithm with Outlier Removal

	Precision	Recall	F1-Score	Support
4	0.00%	0.00%	0.00%	15
5	68.42%	72.63%	70.46%	179
6	61.00%	68.16%	64.38%	179
7	65.22%	51.72%	57.69%	58
8	0.00%	0.00%	0.00%	5
Accuracy	64.68%	64.68%	64.68%	64.68%
Macro Avg	38.93%	38.50%	38.51%	436
Weighted Avg	61.81%	64.68%	63.03%	436

Binary Classification Random Forest Algorithm

	Precision	Recall	F1-Score	Support
< 6	77.29%	75.12%	76.19%	213
>= 6	80.59%	82.40%	81.48%	267
Accuracy	79.17%	79.17%	79.17%	79.17%
Macro Avg	78.94%	78.76%	78.84%	480
Weighted Avg	79.13%	79.17%	79.13%	480

Confusion Matrices

The confusion matrices reflect the classification reports in that the matched true values are highest for 5, 6, and 7 scores in the multiclass versions since the true positive values for them greatly outperform the other scores. The matrix shows a better picture of how the algorithm misclassified the scores in general, and most were within +/- 1 of the true score. The scores where this isn't true are wines that were predicted as score 4 and score 8. This is also an area where there is a significant difference seen by the removal of outliers likely due to the nature of the quality score distribution. Neither version predicted any score 4s correctly, however the version with outlier removal classified significantly more true 5s as a 4, and significantly less 6s as a 4. The opposite was true for wines predicted as an 8.

In this case, I would base outlier removal on whether having incorrect classifications close to the top scores or the bottom scores matter with regards to 4 and 8. If one would prefer incorrectly classified score 4s as score 5, remove outliers, and vice versa. If one would prefer incorrectly classified score 8s as score 7, keep outliers. I don't think outlier removal matters at all for score 3, and if one wants any correctly scored 8, don't remove outliers.

Again, the binary version outperforms the multiclass for the reasons stated above. I would use this in instances where the actual score isn't needed and one only needs to know whether a wine is good (>= 6) or bad (<6). This could be done with any threshold though.

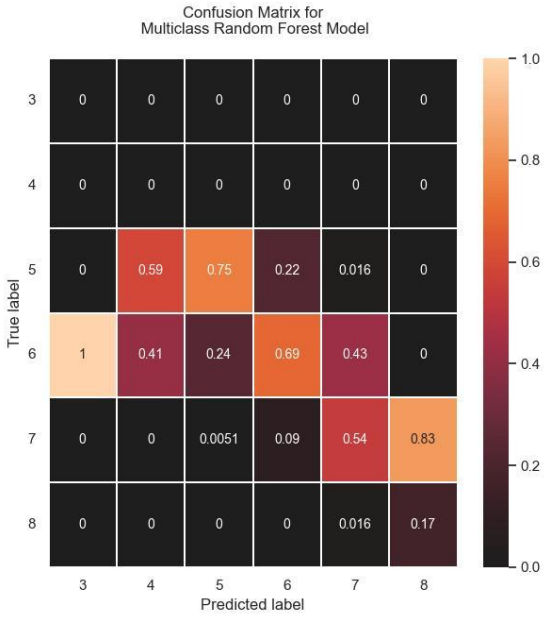
Heatmaps for these matrices can be found on the next page.

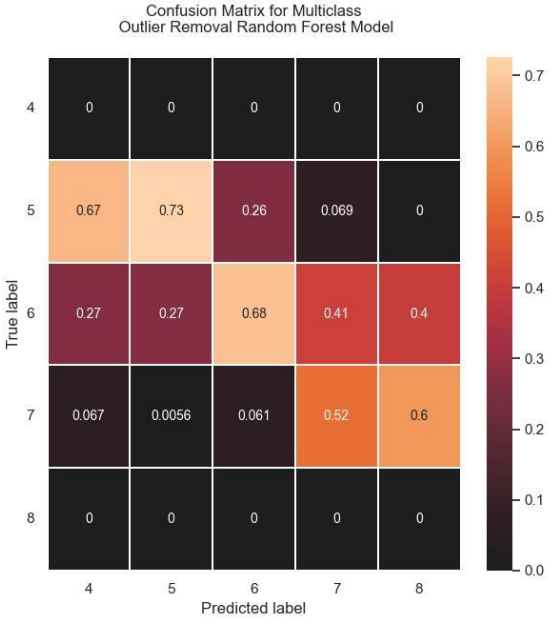
Confusion Matrices

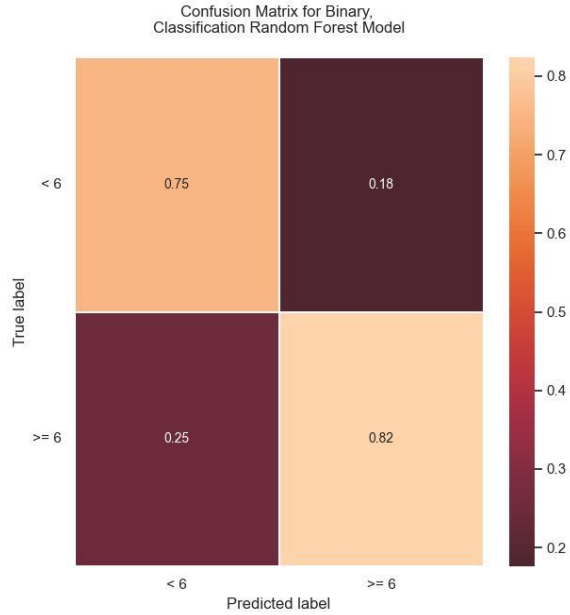
Multiclass		Predicted Score					
		3	4	5	6	7	8
True Score	3	0%	0%	0%	0%	0%	0%
	4	0%	0%	0%	0%	0%	0%
	5	0%	58.82%	75.38%	22.00%	1.64%	0%
	6	100%	41.18%	24.10%	69.00%	42.62%	0%
	7	0%	0%	0.51%	9.00%	54.10%	83.33%
	8	0%	0%	0%	0%	1.64%	16.67%

Multiclass Outlier		Predicted Score				
		4	5	6	7	8
True Score	4	0%	0%	0%	0%	0%
	5	66.67%	72.63%	25.70%	6.90%	0%
	6	26.67%	26.82%	68.16%	41.38%	40.00%
	7	6.67%	0.56%	6.15%	51.72%	60.00%
	8	0%	0%	0%	0%	0%

Binary Class		Predicted Score	
		< 6	>= 6
True Score	< 6	75.12%	17.60%
	>= 6	24.88%	82.40%


Confusion Matrix for
Multiclass Random Forest Model
True label
Predicted label
3 4 5 6 7 8
0 0 0 0 0 0
0 0 0 0 0 0
0 0.59 0.75 0.22 0.016 0
1 0.41 0.24 0.69 0.43 0
0 0 0.0051 0.09 0.54 0.83
0 0 0 0 0.016 0.17
1.0 0.8 0.6 0.4 0.2 0.0


Confusion Matrix for Multiclass
Outlier Removal Random Forest Model
True label
Predicted label
4 5 6 7 8
0 0 0 0 0
0.67 0.73 0.26 0.069 0
0.27 0.27 0.68 0.41 0.4
0.067 0.0056 0.061 0.52 0.6
0 0 0 0 0
0.7 0.6 0.5 0.4 0.3 0.2 0.1 0.0


Confusion Matrix for Binary,
Classification Random Forest Model
True label
Predicted label
< 6 >= 6
0.75 0.18
0.25 0.82
0.8 0.7 0.6 0.5 0.4 0.3 0.2

Python Code Blocks

The full PyCharm project can be found at https://github.com/lt675/IS665_Final_Project

Hyperparameter Tuning

```
# Importing the required libraries
import matplotlib
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
matplotlib.use('TkAgg')

# Read csv files
df = pd.read_csv('winequality-red.csv')

# Define independent variables by removing quality
X = df.drop('quality', axis=1)

# Define dependent variable or label
y = df['quality']

# Split data into training and test sets
# random_state value generally used is Hitch-hikers reference so keep it
there
# Keep random_state so we can get the same results from the train/test splits

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7,
random_state=42)

# Fit the model
# Natural criterion is gini index. Leave it as such
# Not much gained by switching to entropy, which is slower than gini
# oob_score stays true to keep out of the bag cross-validation method
# Not a large dataset so keeping oob_score should be fine
# Keep n_jobs at -1. Let it use all the powers

classifier_rf = RandomForestClassifier(random_state=42, n_jobs=-1,
oob_score=True)
classifier_rf.fit(X_train, y_train)

# Checking the oob score
print(classifier_rf.oob_score_)

# Show current parameters
print(classifier_rf.get_params())

'''
print(classifier_rf.oob_score_)
0.6863270777479893

print(classifier_rf.get_params())
{'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion':
```

```
'gini', 'max_depth': None,
 'max_features': 'sqrt', 'max_leaf_nodes': None, 'max_samples': None,
'min_impurity_decrease': 0.0,
  'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf':
0.0, 'n_estimators': 100,
   'n_jobs': -1, 'oob_score': True, 'random_state': 42, 'verbose': 0,
'warm_start': False}
'''

# Hyperparameter tuning
rf = RandomForestClassifier(random_state=42, n_jobs=-1, oob_score=True)

params = {
    'max_depth': [2, 3, 5, 10, 20, 40, 50],
    'min_samples_leaf': [1, 2, 5, 10, 20, 50],
    'n_estimators': [10, 25, 30, 50, 100, 200, 350, 500, 750, 1000],
    'max_features': [5, 7, 9, 'sqrt', 'log2'],
    'min_samples_split': [2, 4, 5, 10]
}

# Instantiate the grid search model
grid_search = GridSearchCV(estimator=rf,
                           param_grid=params,
                           cv=4,
                           n_jobs=-1, verbose=3, scoring="accuracy")

'''Fitting 4 folds for each of 8400 candidates, totaling 33600 fits
This took a long time to compile. In the future, prune the parameters,
 lower the cv, or get a better computer'''

grid_search.fit(X_train, y_train)

print(grid_search.best_score_)

rf_best = grid_search.best_estimator_
print(rf_best)

# Show current parameters
print(rf_best.get_params())

'''#Below are the results from the Hyperparameter Tuning

print(grid_search.best_score_)
0.6720718125960061

print(rf_best)
RandomForestClassifier(max_depth=40, n_estimators=30, n_jobs=-1,
oob_score=True,
                       random_state=42)

print(rf_best.get_params())
{'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion':
'gini', 'max_depth': 40,
'max_features': 'sqrt', 'max_leaf_nodes': None, 'max_samples': None,
'min_impurity_decrease': 0.0,
'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf':
0.0, 'n_estimators': 30,
```

```
'n_jobs': -1, 'oob_score': True, 'random_state': 42, 'verbose': 0,
'warm_start': False}'''
```

Multiclass Random Forest

```
# Importing the required libraries
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import pandas
import pandas as pd
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
from sklearn.model_selection import train_test_split
matplotlib.use('TkAgg')

# Read csv files
df = pd.read_csv('../winequality-red.csv')

# Define independent variables by removing quality
X = df.drop('quality', axis=1)

# Define dependent variable or label
y = df['quality']

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7,
random_state=42)

# Define parameters for random forest using the ones from hyperparameter
tuning
rf2 = RandomForestClassifier(max_depth=40, n_estimators=30, n_jobs=-1,
oob_score=True,
                             random_state=42)

# Fit model
rf2.fit(X_train, y_train)

# Make predictions for the test set
y_pred_test = rf2.predict(X_test)

Acc = (accuracy_score(y_test, y_pred_test))
obscore = rf2.oob_score_
f = open('RF_Multi_Accuracy_Scores.txt', 'w+')
print('The Accuracy score for the multiclass model is', format(Acc, '.2%'),
file=f)
print('The out of bag score for the multiclass model is', format(obscore,
'.2%'), file=f)
f.close()

# Get and reshape confusion matrix data
matrix = confusion_matrix(y_test, y_pred_test)
```

```
matrix = matrix.astype('float') / matrix.sum(axis=1)[:, np.newaxis]

# Create .csv version of confusion matrix
class_names = ['3', '4', '5', '6', '7', '8']
mcsv = pandas.DataFrame(matrix, index=class_names,
columns=class_names).transpose()
mcsv.to_csv("RF_Multi_Confusion_Matrix.csv")


# Build a plot for a confusion matrix heatmap
plt.figure(figsize=(7, 7))
sns.set(font_scale=1)
sns.color_palette("colorblind", as_cmap=True)
sns.heatmap(mcsv, center=0, annot=True, annot_kws={'size': 10},
            linewidths=0.2)

# Add labels to the plot
plt.xticks(rotation=0)
plt.yticks(rotation=0)
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix for\n Multiclass Random Forest Model\n')

#Save the Heatmap
plt.savefig('RF_Multi_CM_Heat.jpeg')

# Create the classification report for test data and predictions
report = (classification_report(y_test, y_pred_test, output_dict=True))
cr = pandas.DataFrame(report).transpose()
cr.to_csv('RF_Multi_Class_Report.csv')
```

Multiclass Random Forest with Outlier Removal

```
# Importing the required libraries
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import pandas
import pandas as pd
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
from scipy import stats
from sklearn.model_selection import train_test_split

matplotlib.use('TkAgg')

# Read csv files
df = pd.read_csv('../winequality-red.csv')

# Find z-scores to determine outliers. IQR method is too time-consuming.
z = np.abs(stats.zscore(df))
```

```
# Remove data points with a z-score greater than 3
# Based on the 68-95-99.7 rule, this should leave us with points that lie
within
# 99.7% of the data since those are within 3 standard deviations of the mean.
df_o = df[(z < 3).all(axis=1)]

# Define independent variables by removing quality
X = df_o.drop(columns='quality')

# Define dependent variable or label
y = df_o['quality']

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7,
random_state=42)

# Define parameters for random forest using the ones from hyperparameter
tuning
rf4 = RandomForestClassifier(max_depth=40, n_estimators=30, n_jobs=-1,
oob_score=True,
                             random_state=42)

# Fit model
rf4.fit(X_train, y_train)

# Make predictions for the test set
y_pred_test = rf4.predict(X_test)

# View accuracy scores
Acc = (accuracy_score(y_test, y_pred_test))
obscore = rf4.oob_score_
f = open('RF_Outlier_Accuracy_Scores.txt', 'w+')
print('The Accuracy score for the multiclass model with outlier removal is',
format(Acc, '.2%'), file=f)
print('The out of bag score for the multiclass model with outlier removal
is', format(obscore, '.2%'), file=f)
f.close()

# Get and reshape confusion matrix data
matrix = confusion_matrix(y_test, y_pred_test)
matrix = matrix.astype('float') / matrix.sum(axis=1)[:, np.newaxis]

# Create .csv version of confusion matrix
class_names = ['4', '5', '6', '7', '8']
mcsv = pandas.DataFrame(matrix, index=class_names,
columns=class_names).transpose()
mcsv.to_csv("RF_Outlier_Confusion_Matrix.csv")

# Build a plot for a confusion matrix heatmap
plt.figure(figsize=(7, 7))
sns.set(font_scale=1)
sns.color_palette("colorblind", as_cmap=True)
sns.heatmap(mcsv, center=0, annot=True, annot_kws={'size': 10},
            linewidths=0.2)

# Add labels to the plot
plt.xticks(rotation=0)
```

```
plt.yticks(rotation=0)
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix for Multiclass\n Outlier Removal Random Forest
Model\n')

# Save the Heatmap
plt.savefig('RF_Outlier_CM_Heat.jpeg')

# Create the classification report for test data and predictions
report = (classification_report(y_test, y_pred_test, output_dict=True))
cr = pandas.DataFrame(report).transpose()
cr.to_csv('RF_Outlier_Class_Report.csv')
```

Binary Classification Random Forest

```

# Importing the required libraries
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import pandas
import pandas as pd
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
from sklearn.model_selection import train_test_split
matplotlib.use('TkAgg')

# Read csv files
df = pd.read_csv('../winequality-red.csv')

# Define binary data split and create new dataframe
ddf = pd.DataFrame()
ddf['class'] = df['quality']
ddf.loc[(ddf['class'] < 6), 'new_class'] = 0
ddf.loc[(ddf['class'] >= 6), 'new_class'] = 1

# Concatenate original and data split dataframes
df_bin = pd.concat((ddf, df), axis=1)

# Remove unnecessary columns
df_bin = df_bin.drop(['class'], axis=1)
df_bin = df_bin.drop(['quality'], axis=1)

# Define independent variables by removing new class column
X = df_bin.drop('new_class', axis=1)

# Define dependent variable or label
y = df_bin['new_class']

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7,
random_state=42)
```

```
# Define parameters for random forest using the ones from hyperparameter
tuning
rf3 = RandomForestClassifier(max_depth=40, n_estimators=30, n_jobs=-1,
oob_score=True,
                             random_state=42)

# Fit model
rf3.fit(X_train, y_train)

# Make predictions for the test set
y_pred_test = rf3.predict(X_test)

# View accuracy scores
Acc = (accuracy_score(y_test, y_pred_test))
obscore = rf3.oob_score_
f = open('RF_Binary_Accuracy_Scores.txt', 'w+')
print('The Accuracy score for the binary classification model is',
format(Acc, '.2%'), file=f)
print('The out of bag score for the binary classification model is',
format(obscore, '.2%'), file=f)
f.close()

# Get and reshape confusion matrix data
matrix = confusion_matrix(y_test, y_pred_test)
matrix = matrix.astype('float') / matrix.sum(axis=1)[:, np.newaxis]

# Create .csv version of confusion matrix
class_names = ['< 6', '>= 6']
mcsv = pandas.DataFrame(matrix, index=class_names,
columns=class_names).transpose()
mcsv.to_csv("RF_Binary_Confusion_Matrix.csv")

# Build a plot for a confusion matrix heatmap
plt.figure(figsize=(7, 7))
sns.set(font_scale=1)
sns.color_palette("colorblind", as_cmap=True)
sns.heatmap(mcsv, center=0, annot=True, annot_kws={'size': 10},
            linewidths=0.2)

# Add labels to the plot
plt.xticks(rotation=0)
plt.yticks(rotation=0)
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix for Binary, \nClassification Random Forest
Model\n')

# Save the Heatmap
plt.savefig('RF_Binary_CM_Heat.jpeg')

# Create the classification report for test data and predictions
report = (classification_report(y_test, y_pred_test, output_dict=True))
cr = pandas.DataFrame(report).transpose()
cr.to_csv('RF_Binary_Class_Report.csv')
```

Decision Tree by Beyzanur Tokar

Decision Tree

This section is dedicated to conducting classification through the application of the Decision Tree algorithm. I've deliberately chosen the Decision Tree algorithm for the analysis and modeling of the red wine dataset. Although its proficiency in handling categorical to categorical datasets, a noteworthy advantage of Decision Trees is their seamless management of a mix of numerical and categorical features, eliminating the need for extensive pre-processing. I'm optimistic that this flexibility will prove effective in our context. Not only does this simplify the modeling process, but it also enhances the algorithm's adaptability to the diverse characteristics inherent in the dataset.

```
# Declare feature vector and target variable ¶
x = WineQualityRed[['fixed acidity',  'citric acid', 'residual sugar', 'sulphates', 'alcohol']]
y = WineQualityRed['quality']
print(x,y)
```

Based on the results of the confusion matrix, I have chosen to analyze the data using the five attributes positively correlated with the outcome. These attributes are Alcohol, sulphates, citric acid, Fixed acidity, and residual sugar.

```
from sklearn.model_selection import train_test_split
```

```
# split X and y into training and testing sets

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3)
```

```
# check the shape of x_train and x_test

x_train.shape, x_test.shape
```

```
((1119, 5), (480, 5))
```

Afterward, I divided the wine quality dataset into two categories: training data, which constitutes 70% of the dataset, and testing data, encompassing the remaining 30%. Consequently, I obtained a training dataset of dimensions (1119, 5) and a testing dataset of dimensions (480, 5).

```
# import DecisionTreeClassifier

from sklearn.tree import DecisionTreeClassifier
```

```
# instantiate the DecisionTreeClassifier model with criterion entropy

dtc = DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=245)


# fit the model
dtc_pred_da = dtc.fit(x_train, y_train)
```

```
dtc_pred = dtc.predict(x_test)
```

```
# print the scores on training and test set

print('Training set score: {:.4f}'.format(dtc.score(x_train, y_train)))

print('Test set score: {:.4f}'.format(dtc.score(x_test, y_test)))
```

```
Training set score: 0.5934
Test set score: 0.5896
```

Upon training the dataset with a decision tree, the achieved accuracy is 59%. I guess it is not a good result.

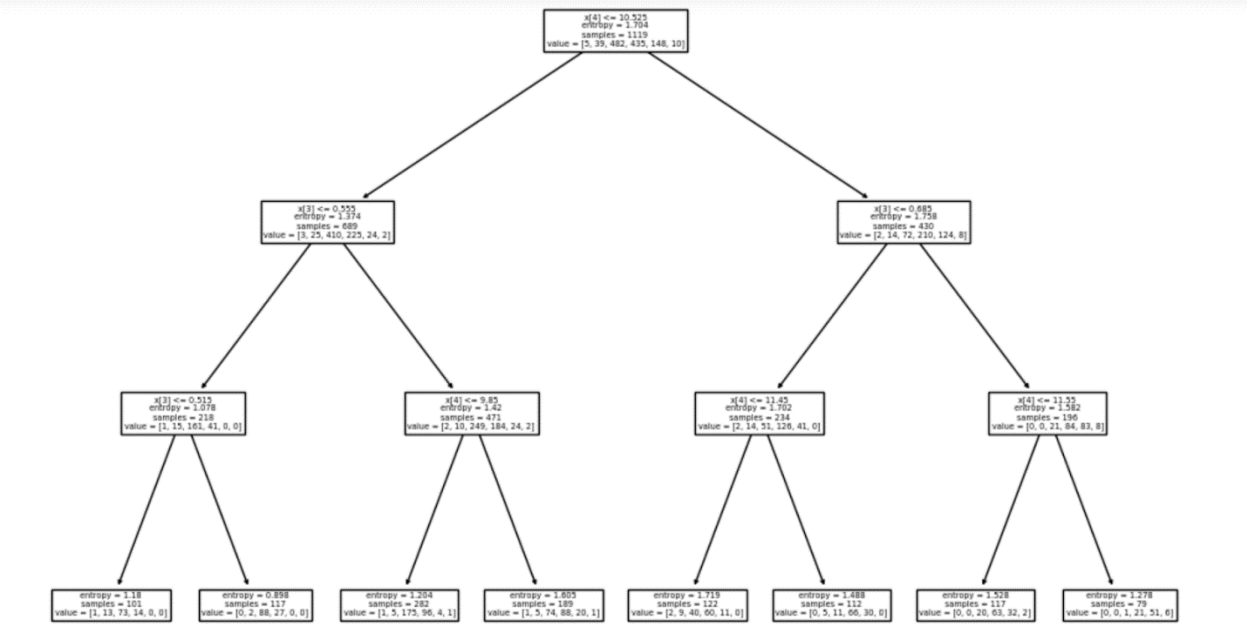
```
pd.DataFrame({
    'actual' : y_test[:10],
    'classification' : dtc_pred[:10],
    })
```

	actual	classification
986	7	6
1313	6	5
261	4	5
1221	6	6
7	7	5
1408	7	7
1356	5	5
176	5	5
1241	5	5
1136	6	6

My critique is centered around the scoring process and the definition of a successful prediction. I am particularly skeptical about the quality of the reported scores, given their modest 59% accuracy. The decision tree method, in particular, achieves correctness only two out of three times, prompting me to question its efficacy.

I suspect that a significant part of the issue lies within the scoring methodology itself, where the treatment of a miss appears to be considered as equivalent to a substantial error.

```
: # draw decision tree
  plt.figure(figsize=(12,8))
  from sklearn import tree
  tree.plot_tree(dtc.fit(x_train, y_train))
```



X[0] ='fixed acidity', X[1]= 'citric acid', X[4] = ‘alcohol’, x[3] = 'sulphates'
At the first node, we see the result of these:

X[4] <= 10.525
entropy = 1.704
samples= 1119

values = [5, 39,482,435,148,10]

- The feature X[4] is labeled as 'alcohol,' it means that the decision tree is making a split based on the value of the 'alcohol' feature. The condition X[4]≤10.525 that the decision tree is specifically looking at the alcohol feature and making a decision based on whether its value is less than or equal to 10.525.
- The entropy at this node is a measure of impurity. A higher entropy indicates more mixed or impure classes in the node. In this condition the most impure class is alcohol.
- 1119 is the total number of samples in the node.
- This represents the distribution of classes or labels in the node after the split. Each number in the array corresponds to the count of samples for a particular class. For example, there are 5 samples of one class, 39 samples of another class, 482 samples of a third class, and so on.

Decision Tree Binary

```
def quality_binominal(cols):
    quality = cols[0]
    if quality >= 6:
        return 1
    else:
        return 0
```

```
WineQualityRed['quality'] = WineQualityRed[['quality']].apply(quality_binominal,axis=1)
```

```
WineQualityRed.head()
```

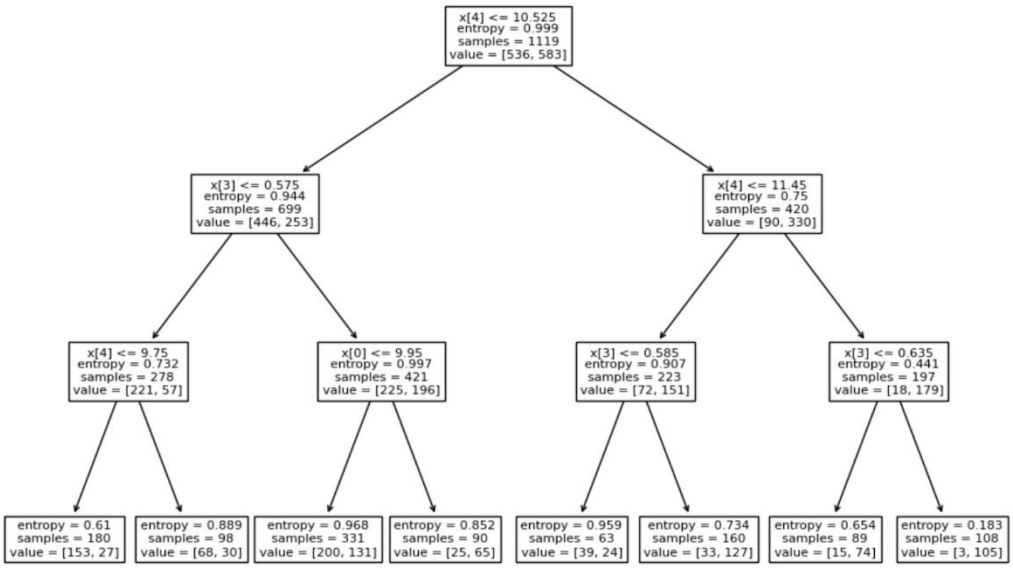
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	0
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	0
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	0
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	1
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	0

```
# print the scores on training and test set

print('Training set score: {:.4f}'.format(dtc.score(x_train, y_train)))

print('Test set score: {:.4f}'.format(dtc.score(x_test, y_test)))
```

```
Training set score: 0.7426
Test set score: 0.7021
```

```
pd.DataFrame({
    'actual' : y_test[:10],
    'classification' : dtc_pred[:10],
    })
```

	actual	classification
1338	0	0
401	1	0
1032	0	0
1007	1	1
1381	0	0
1491	0	0
696	1	0
871	0	0
365	1	1
309	1	0



As evident, each node contains two values, indicating whether the outcome for quality is categorized as 'true' or 'false.'

```
from sklearn.metrics import ConfusionMatrixDisplay
```

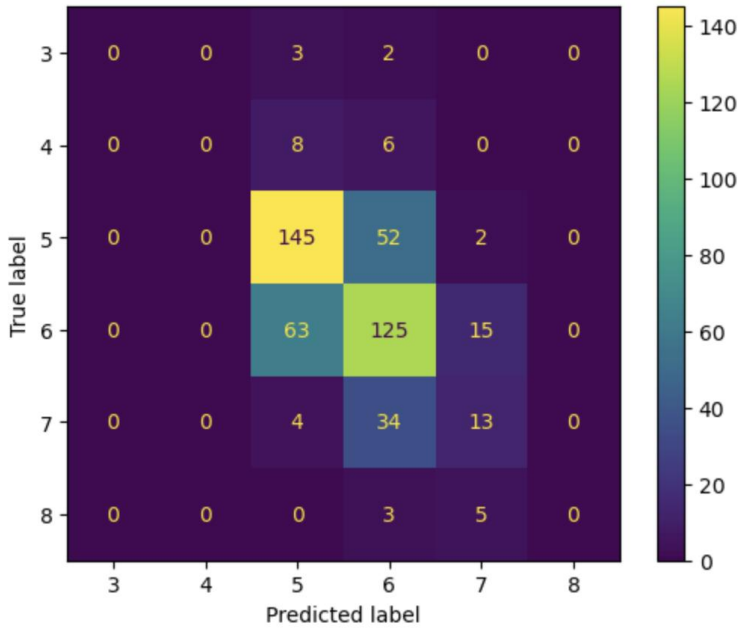
```
ConfusionMatrixDisplay.from_estimator(
    dtc_pred_da,
    x_test,
    y_test
)
```

Conclusion

Decision Tree

```
from sklearn.metrics import ConfusionMatrixDisplay
```

```
ConfusionMatrixDisplay.from_estimator(
    dtc_pred_da,
    x_test,
    y_test
)
```



- This graph illustrates the prediction of wine quality, with labels ranging from 3 to 8 representing the quality levels. The x-axis displays the predicted labels, while the y-axis indicates the true labels. This visual representation allows us to discern both correct and incorrect predictions, providing insights into the accuracy of our model.

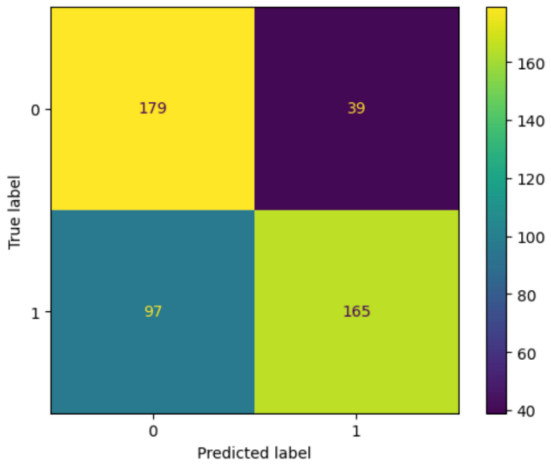
- Typically, a confusion matrix involves four categories, but in our case, with more than two categories (0,1), we are dealing with a 6x6 matrix to comprehensively represent the multiple categories we have in our dataset.

```
from sklearn.metrics import classification_report

print(classification_report(y_test, dtc_pred))
```

```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         5
           4       0.00      0.00      0.00        14
           5       0.65      0.73      0.69       199
           6       0.56      0.62      0.59       203
           7       0.37      0.25      0.30        51
           8       0.00      0.00      0.00         8

    accuracy                           0.59       480
   macro avg       0.26      0.27      0.26       480
weighted avg       0.55      0.59      0.57       480
```

- The model seems to perform well on classes 5 and 6, as indicated by relatively high precision, recall, and F1-score.
- Classes 3, 4, and 8 have very low precision, recall, and F1-score, suggesting poor performance on these classes.
- Class 7 has moderate precision and recall, with an F1-score of 0.30.
- The overall accuracy of the model is 59%, but it's important to consider the class distribution, especially if it's imbalanced.
- F1-scores for each class (e.g., 0.69 for class 5, 0.59 for class 6) provide a balanced measure of precision and recall.
- Higher F1-scores indicate better overall performance in that class.

Decision Tree Binominal




```
from sklearn.metrics import classification_report

print(classification_report(y_test, dtc_pred))
```

	precision	recall	f1-score	support
0	0.61	0.84	0.71	208
1	0.83	0.60	0.70	272
accuracy			0.70	480
macro avg	0.72	0.72	0.70	480
weighted avg	0.74	0.70	0.70	480

For Class 0: The precision of 0.61 indicates that when the model predicts class 0, it is correct 61% of the time. The recall of 0.84 suggests that the model is able to capture 84% of the actual instances of class 0. The F1-score of 0.71 provides a balanced measure of precision and recall for class 0. With a support of 208, class 0 has a relatively moderate number of instances.

For Class 1: The precision of 0.83 indicates a high level of correctness when predicting class 1. The recall of 0.60 suggests that the model captures 60% of the actual instances of class 1. The F1-score of 0.71 provides a balanced measure of precision and recall for class 1. Class 1 has a higher support of 272 compared to class 0.

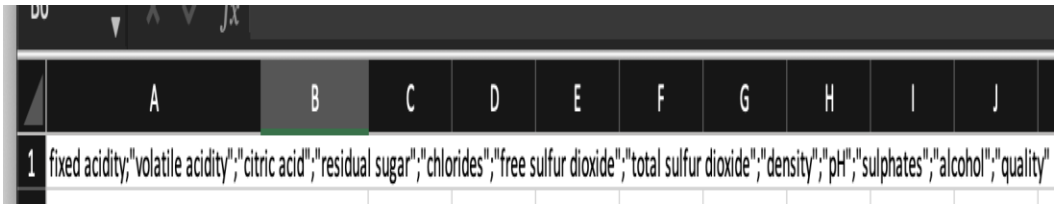
The overall accuracy of the model on the entire dataset is 70%.

The model seems to perform reasonably well, with good precision, recall, and F1-score for both classes.

Logistic Regression (supervised machine learning algorithm) by Zhenya Venglinski

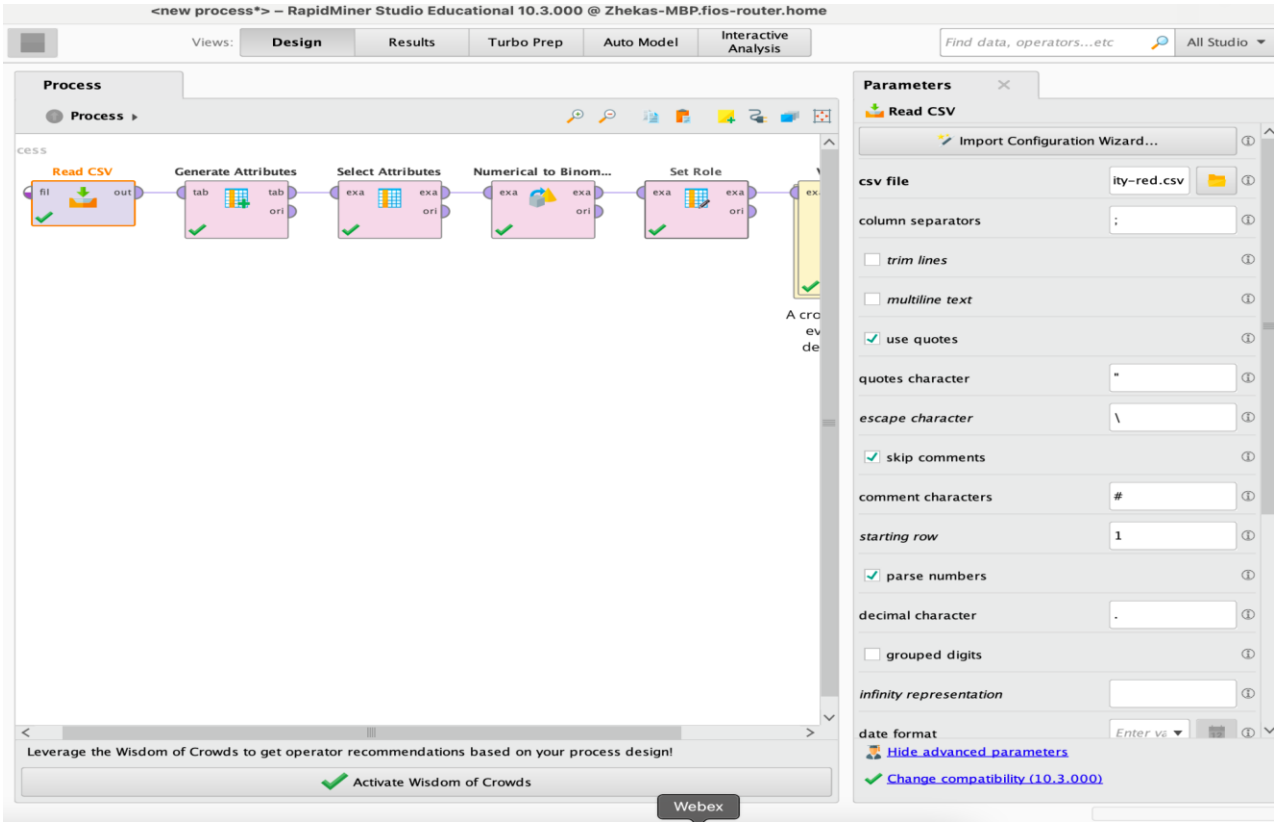
In my analysis I used logistic regression, which is considered to be a supervised machine learning algorithm. I believe logistic regression was another suitable algorithm to be utilized to predict wine quality based on its physicochemical properties through analysis of relationships between different variables. The outcome of a logistic regression will be a binary outcome with data from past test results in place. Let's look at the results of processing our dataset in Rapidminor using the logistic regression within the validation parameter.

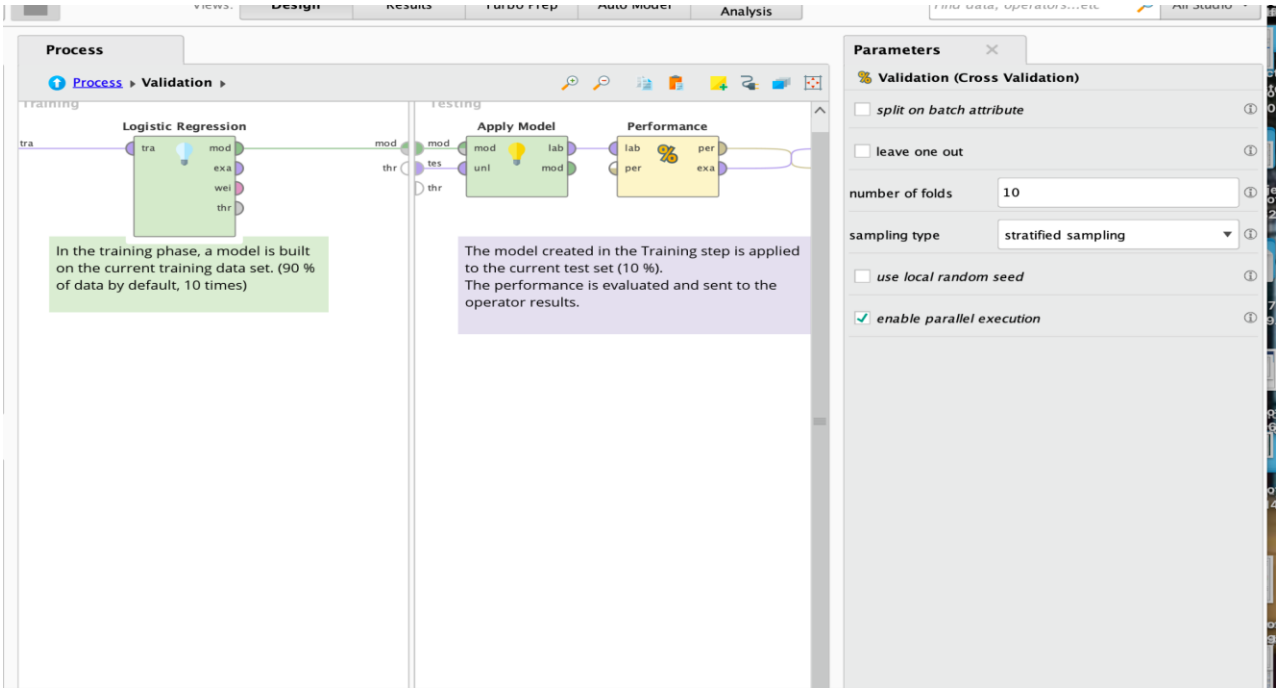
The dataset that we chose to analyze consists of wine samples with specific attributes like:citric acid, residual sugar pH and others(see the screenshot below).



The goal of this report is to apply a supervised learning algorithm which in this case is logistic regression, and predict wine quality through examination of report results accordingly. By applying different algorithms like kNN, logistic regression and others, assist us in targeting a variety of correlation and patterns of our dataset, which deepens our understanding of the role of specific attributes on the wine quality and improves overall wine quality prediction based on its chemical composition

Process Workflow and Parameters Setup

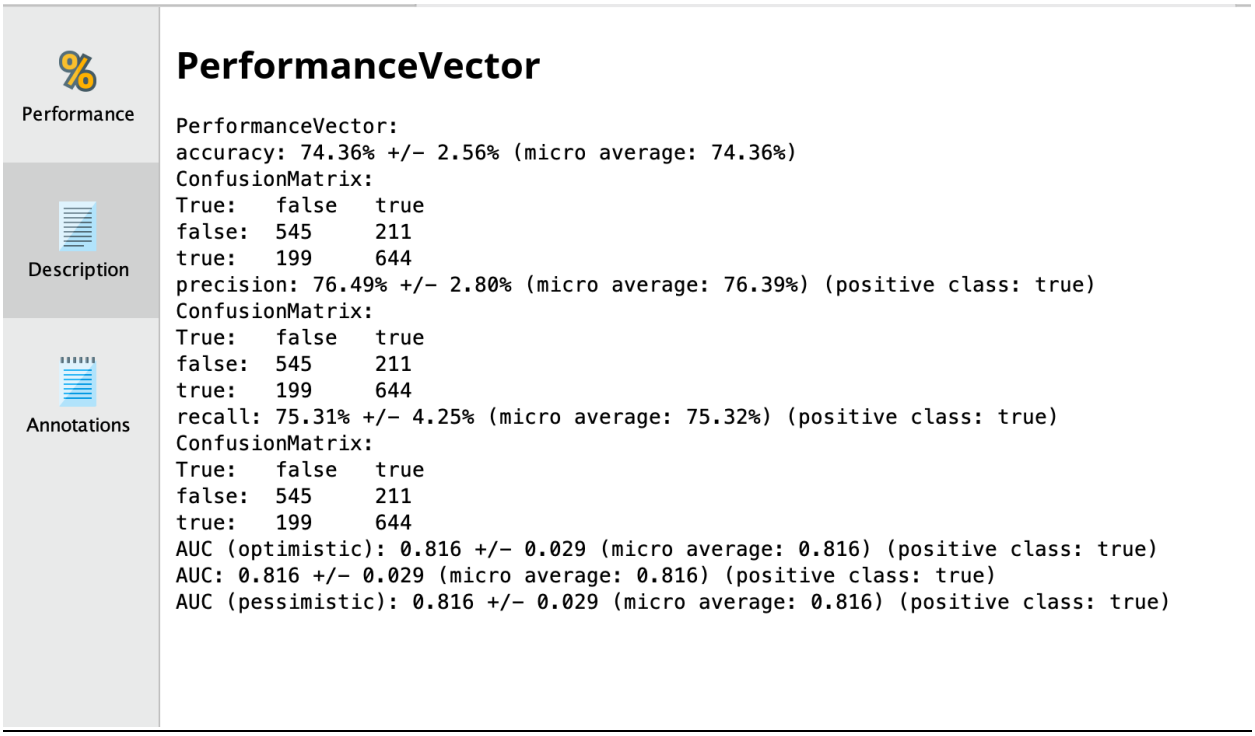




Above you can see the workflow with the RapidMiner with the following operator in place. When setting up a workflow its important to remember that order matters otherwise your analyzing either will not work/flow properly or give you skewed results. Most of the time RapidMiner is smart enough to detect setup errors.

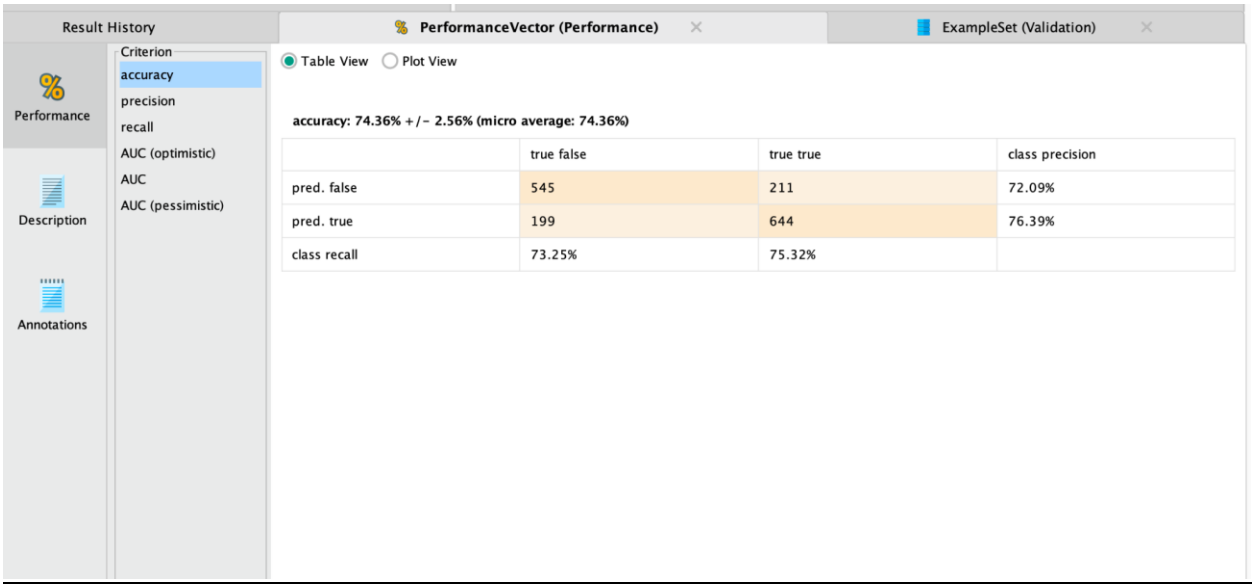
Results

Confusion Matrix

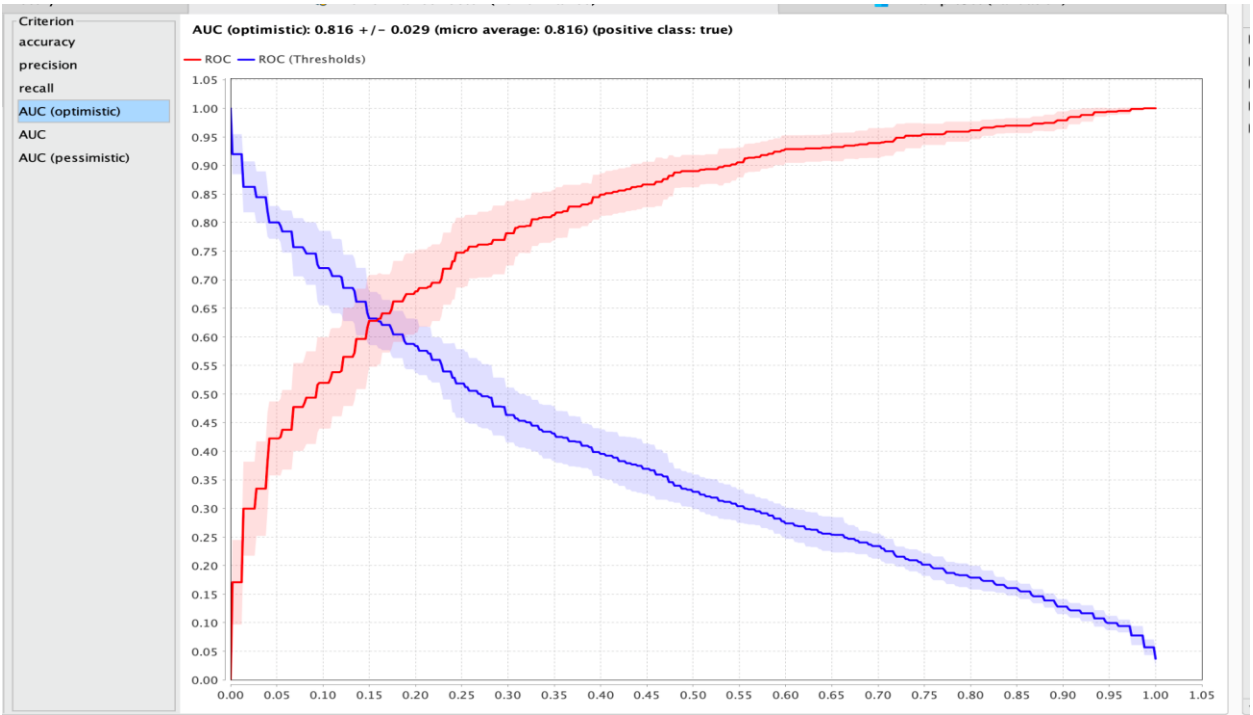


The values above are generated from the performance operator. It shows the overall summary of the numbers for accuracy, precision, AUC values along with the confusion matrix. Confusion matrix reports true and false values which helps to measure the factors affecting the quality of wine, basically how well the model is performing, which overall aids in making informed future decisions.

Performance



Above is the confusion matrix see the overall weighted average of the accuracy for true and false values. This shows what types of mistakes my model is making. True validation participation that we used in 72.09 percent of the data we have 545 and 199 = 744 data points turned out to be true false with the class recall which is the percentage of accuracy just for the true false points to be 73.25 percent. We can also see that 199 data points were misinterpreted in our dataset analysis. If we look at the other column, we have 211 and 644 values, where 644 were predicted as true and 211 were predicted as false making them being misclassified. We can also see the overall accuracy of 74.36 percent which is calculated through correct prediction/total of the record in regards to prediction of y=1. Its important to look at both percentages in the confusion matrix, the overall and the class recall percentages, because sometimes weighted averages can be skewed. Here we have percentages close to each other for both classes and overall accuracy value with the +/- 2.56.



AUC (optimistic) graph: displays ROC Curve with the AUC optimistic value of 0.816 +/- 0.029. The ROC curve is a plot of true positive and false positive rates in different individual validation tests. The AUC value of 0.816 is an indicator of a good fit/model.

Logistic Regression Model

ExampleSet (Set Role) | Logistic Regression Model (Logistic Regression)

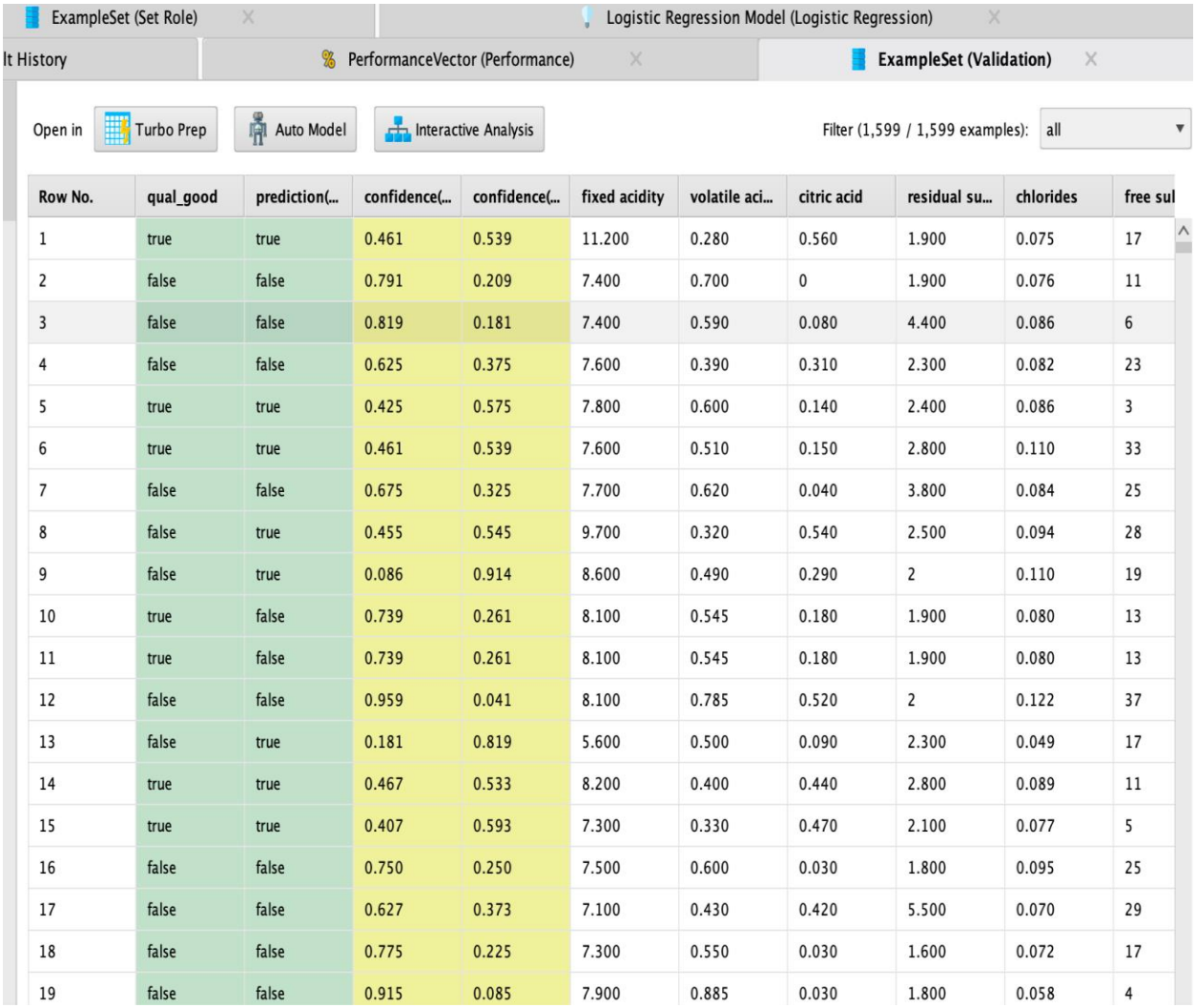
History | PerformanceVector (Performance) | ExampleSet (Validation)

Attribute	Coefficient	Std. Coefficient	Std. Error	z-Value	p-Value
fixed acidity	0.136	0.236	0.098	1.378	0.168
volatile acidity	-3.281	-0.587	0.488	-6.719	0.000
citric acid	-1.273	-0.248	0.563	-2.263	0.024
residual sugar	0.055	0.078	0.054	1.027	0.304
chlorides	-3.918	-0.184	1.569	-2.496	0.013
free sulfur dioxide	0.022	0.234	0.008	2.709	0.007
total sulfur dioxide	-0.016	-0.540	0.003	-5.694	0.000
density	-50.770	-0.096	81.153	-0.626	0.532
pH	-0.382	-0.059	0.720	-0.531	0.596
sulphates	2.795	0.474	0.452	6.182	0.000
alcohol	0.867	0.924	0.104	8.319	0
Intercept	42.794	0.240	79.478	0.538	0.590

The screenshot above shows different attributes and its corresponding numbers for coefficients, z - value and p value. What I would like to highlight is the p - value. P value is a statistical measurement used to validate hypothesis against our observed dataset. As you can see in the above table it appears that there are four attributes that we can consider statistically significant and null should be rejected which are volatile acidity, total siflur dioxide, sulphates and alcohol.

It is also important to point out the z-value which indicates how far your value is to the mean. The negative z-values indicate values being below the mean (average) which appear to be volatile acidity, citric acid, chlorides total sulfur dioxide, density and even pH.

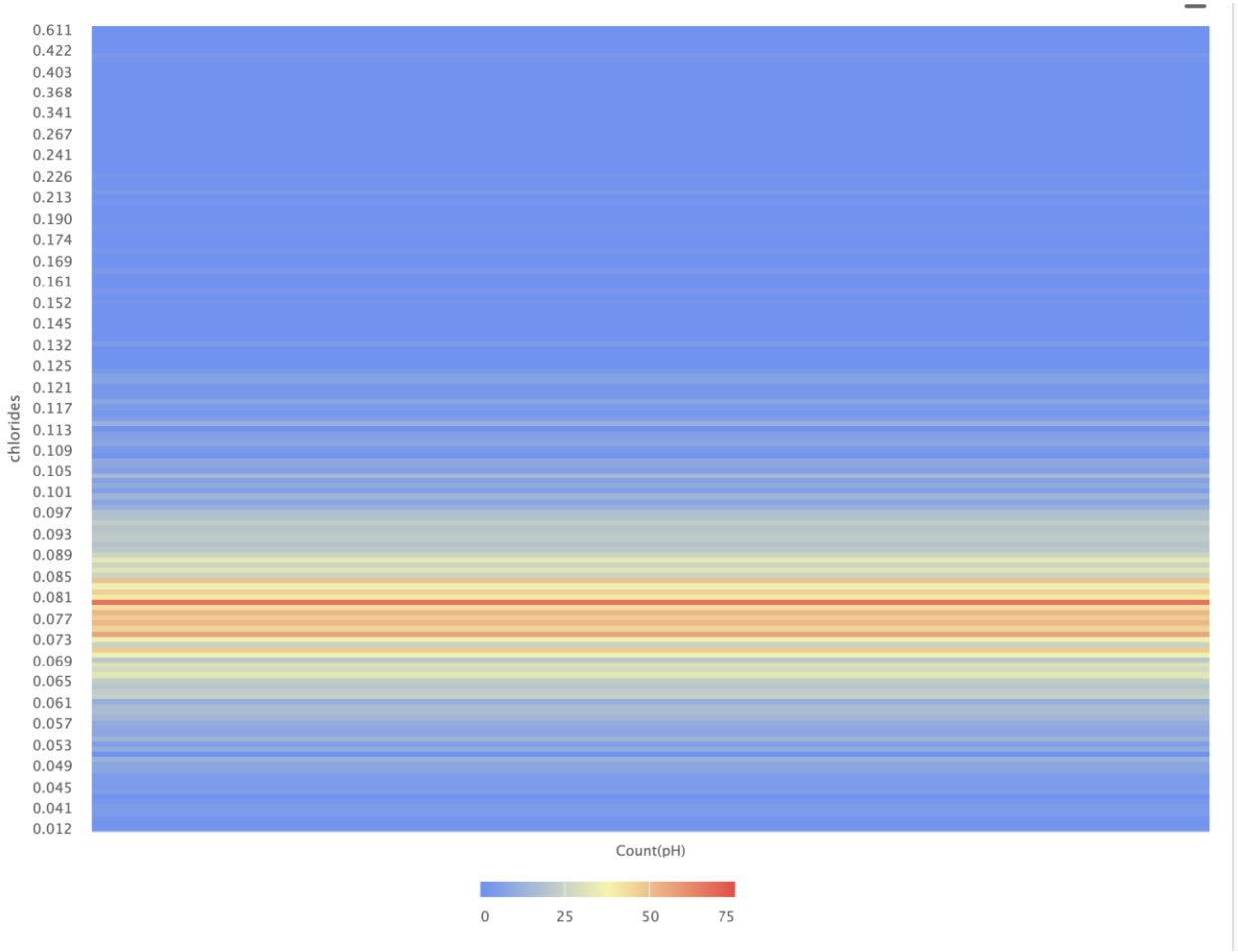
Example Set (Validation)



Row No.	qual_good	prediction(...	confidence(...	confidence(...	fixed acidity	volatile aci...	citric acid	residual su...	chlorides	free sul
1	true	true	0.461	0.539	11.200	0.280	0.560	1.900	0.075	17
2	false	false	0.791	0.209	7.400	0.700	0	1.900	0.076	11
3	false	false	0.819	0.181	7.400	0.590	0.080	4.400	0.086	6
4	false	false	0.625	0.375	7.600	0.390	0.310	2.300	0.082	23
5	true	true	0.425	0.575	7.800	0.600	0.140	2.400	0.086	3
6	true	true	0.461	0.539	7.600	0.510	0.150	2.800	0.110	33
7	false	false	0.675	0.325	7.700	0.620	0.040	3.800	0.084	25
8	false	true	0.455	0.545	9.700	0.320	0.540	2.500	0.094	28
9	false	true	0.086	0.914	8.600	0.490	0.290	2	0.110	19
10	true	false	0.739	0.261	8.100	0.545	0.180	1.900	0.080	13
11	true	false	0.739	0.261	8.100	0.545	0.180	1.900	0.080	13
12	false	false	0.959	0.041	8.100	0.785	0.520	2	0.122	37
13	false	true	0.181	0.819	5.600	0.500	0.090	2.300	0.049	17
14	true	true	0.467	0.533	8.200	0.400	0.440	2.800	0.089	11
15	true	true	0.407	0.593	7.300	0.330	0.470	2.100	0.077	5
16	false	false	0.750	0.250	7.500	0.600	0.030	1.800	0.095	25
17	false	false	0.627	0.373	7.100	0.430	0.420	5.500	0.070	29
18	false	false	0.775	0.225	7.300	0.550	0.030	1.600	0.072	17
19	false	false	0.915	0.085	7.900	0.885	0.030	1.800	0.058	4

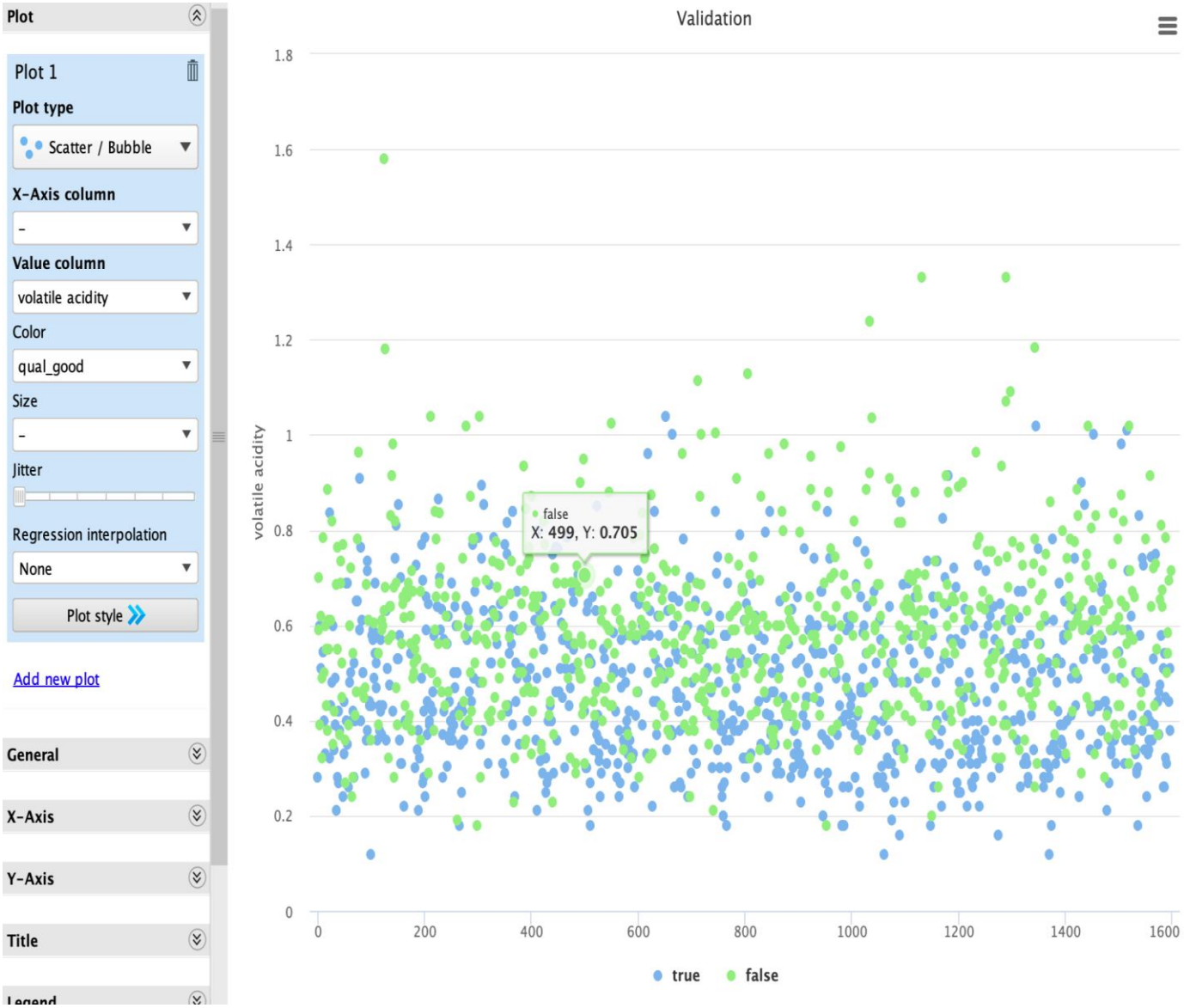
Above you can see the output of the probability of the targeted variable. The green column is the predicted y, with prediction based off of confidence values. Here we have two probabilities for wine quality in the confidence rows as true and false, and on the left side we have a true/false label that records the proper label according to a certain cutoff of probability. For example, the confidence values of 0.461 and 0.539 are considered to be true for both indicating positive correlation of wine quality for that particular threshold of analysis. In summary the higher the confidence number the further the distance of our data point from the probability cutoff.

Heatmap



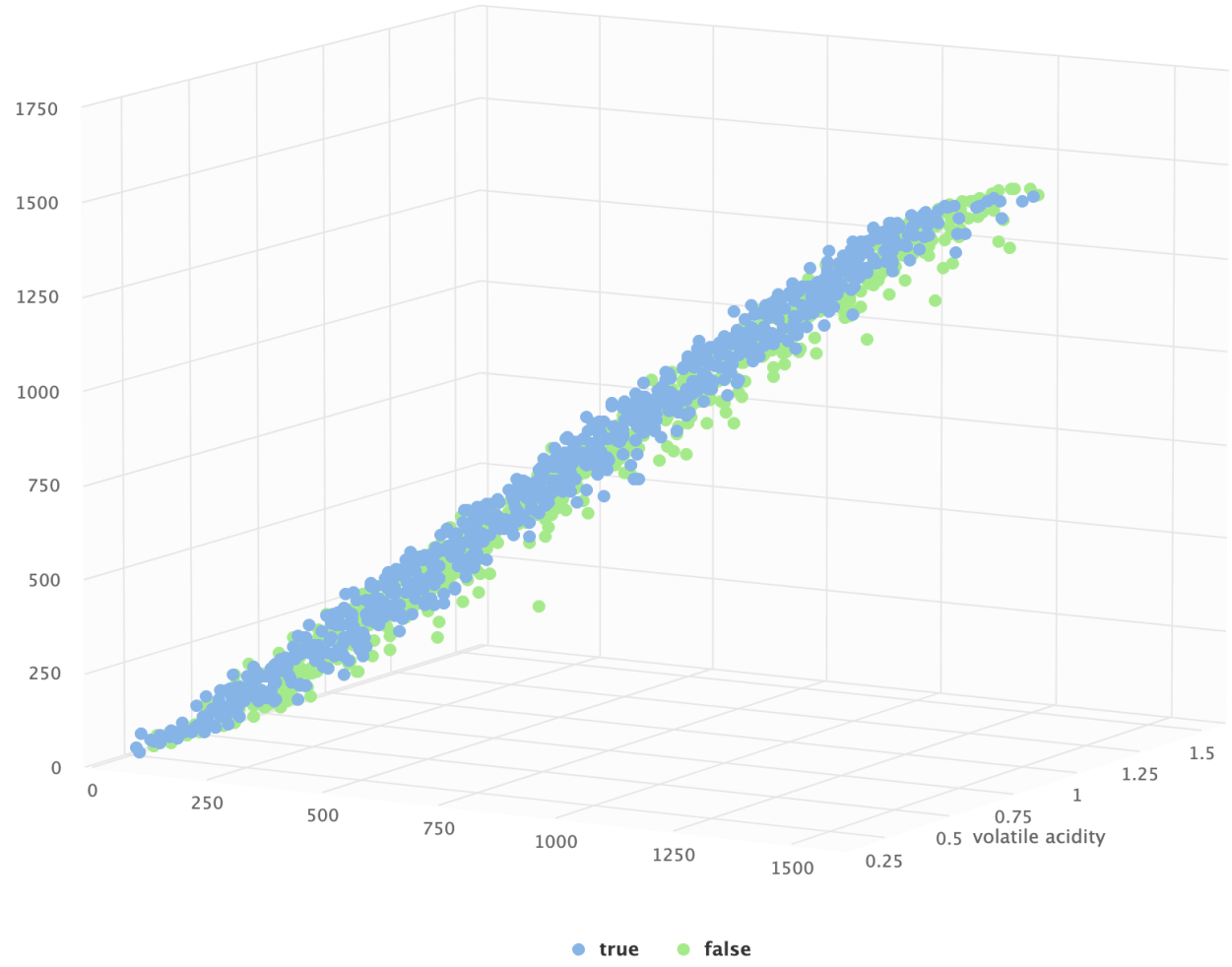
Above you can observe the heatmap. Purpose of the heatmap is graphical representation data by color. It helps us to visualize and direct our attention towards areas that matter most. If we look at the heatmap above we can clearly see where pH is greater compared to the rest if the chlorine level which appears to be much greater in the 0.069 - 0.085 ranges.

Scatter



Looking at the scatter plots above you can see that centroids are placed in a linear pattern with data points being close to each other, therefore close in value. The predictive decision boundary (fine line between blue and green points) is not linear here for a volatile acidity scatter(not linear) plot with blue points indicating that we have some errors, predictions that turn out to be not as expected.

3D Scatter



Here we can see our scatter plot, data points from the zoomed out view/perspective, which is a great representation of the strong, positive and linear correlation of our data.

Conclusion:

I believe that representing our dataset through Logistic Regression was a great fit. Logistic regression is a statistical technique where a dependent variable is to be predicted and dependable on the independent variable which are the attributes like citric acid, residual sugar pH and others. What sets linear regression apart from a regular linear regression is the dependable variable. In linear regressions the dependent variable usually continues, whereas it is continuous in a logistic regression. Results of the report could be highly beneficial not just for customer information, it could be also useful for marketing strategies and lab studies. Overall, the results of this analysis found to be consistent, strong with positive and linear correlation.

Part IV. Conclusion

Best Evaluation Metrics

Accuracy would feel like the obvious choice to evaluate a model, however this is not the case, especially with an imbalanced dataset. Since it takes into account all of the correctly predicted values, those features with larger numbers of cases will be over-represented. This introduces bias into the measurement.

Precision measures the accuracy of positive predictions. So if a wine is classified as a 3, it measures whether that 3 is truly a 3. Recall measures the accuracy of whether a 3 was actually classified as a 3.

If the cost or consequences of misclassifying a wine is high (e.g., financial loss or customer dissatisfaction), precision becomes crucial. If misclassifying a wine is more problematic than incorrectly classifying one, recall becomes important.

The importance of precision and recall depends on the specific goals of the business. For example, if the goal is to recommend only the highest quality wines to customers, you may prioritize precision. If the goal is to ensure that no high-quality wines are missed, you may prioritize recall.

Since the F1 score takes both precision and recall into account, it may be the best metric in general since there are many different reasons you'd want to classify wine quality.

Algorithm Comparison

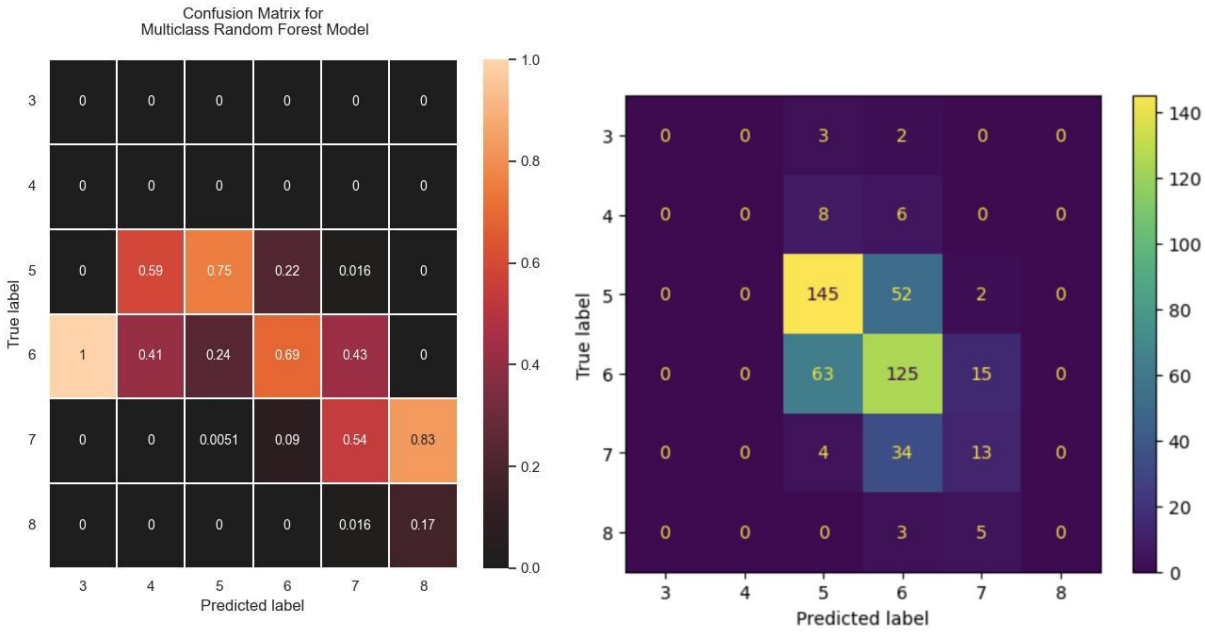
Multiclassification Analysis

The two algorithms used for multiclassification analysis were Decision Tree and Random Forest. Random Forest did outperform Decision Tree in all metrics, which is expected due to Random Forest being an aggregation of Decision Trees.

Classification Reports				
Multiclass Random Forest Algorithm				
	Precision	Recall	F1-Score	Support
3	0%	0%	0%	1
4	0%	0%	0%	17
5	72.77%	75.38%	74.06%	195
6	63.01%	69.00%	65.87%	200
7	57.89%	54.10%	55.93%	61
8	50%	16.67%	25.00%	6
Accuracy	66.46%	66.46%	66.46%	66.46%
Macro Avg	40.61%	35.86%	36.81%	480
Weighted Avg	63.80%	66.46%	64.95%	480

Decision Tree Algorithm				
	Precision	Recall	F1-Score	Support
3	0%	0%	0%	5
4	0%	0%	0%	14
5	65.00%	73.00%	69.00%	199
6	56.00%	62.00%	59.00%	203
7	37.00%	25.00%	30%	51
8	0%	0%	0%	8
Accuracy	57.00%	57.00%	57.00%	57.00%
Macro Avg	27.00%	27.00%	27.00%	480
Weighted Avg	54.00%	57.00%	55.00%	480

If exact quality scores are needed overall, Random Forest would be the better algorithm to use. Something else to consider though is the misclassification of wines. When looking at the confusion matrices, Decision Tree misclassifications align closer with the true score. Therefore, if one would like the misclassified wines to be closer to their true score, it would make more sense to use Decision Tree even though its general performance isn't as good.



Binary Classification Analysis

Four algorithms were used for the classification of red wine using a binary output. These are Logistic Regression, Decision Tree, k-NN, and Random Forest. All four used a threshold of 6 for classification, and the groups were wines with a quality score below 6 and those with a quality score of 6 and above.

Random Forest Algorithm:
It achieves a balanced performance with high precision, recall, and F1-scores for both classes. The overall accuracy is 79.17%. Both macro and weighted averages are consistent and they reflect a stable model across classes. This algorithm reliably classifies wines based on quality thresholds.

Decision Tree Algorithm:
It displays imbalanced performance, particularly for the ">= 6" class with the 83.00% precision and 61.00% recall rates. The overall accuracy is 71.00%, and there's a noticeable difference in performance between the classes. Macro-average shows a slight imbalance, and weighted averages are consistent but affected by the imbalanced recall.

Logistic Regression Algorithm:
It shows balanced performance with reasonably high precision, recall, and F1-scores for both classes. The overall accuracy is 74.36%. Macro and weighted averages are consistent, suggesting reliability across both classes. Similarly to Random Forest, it reliably classifies wines based on the quality thresholds.

k-NN Algorithm:
Similarly to Logistic Regression and Random Forest, it shows fairly balanced performance. The accuracy score is 81.68%. Wine quality classification can be considered reliable using this algorithm since the weighted and macro averages are consistent.

Considering that both classes (>=6 and <6) are equally important, the k-NN Algorithm looks like to be the most suitable among the four. It demonstrates a balanced performance with high precision, recall, and F1-scores for both classes and resulting in an overall accuracy of 81.68%. The Decision Tree Algorithm shows some imbalance, especially in recall for the ">= 6" class, while the Logistic Regression and Random Forest algorithms perform well but with slightly lower metrics compared to k-NN. Therefore, the k-NN Algorithm is the best algorithm for this scenario.

Classification Reports

Random Forest Algorithm

	Precision	Recall	F1-Score	Support
< 6	77.29%	75.12%	76.19%	213
>= 6	80.59%	82.40%	81.48%	267
Accuracy	79.17%	79.17%	79.17%	
Macro Avg	78.94%	78.76%	78.84%	480
Weighted Avg	79.13%	79.17%	79.13%	480

Decision Tree Algorithm

	Precision	Recall	F1-Score	Support
< 6	64.00%	84.00%	73.00%	216
>= 6	83.00%	61.00%	70.00%	264
Accuracy	71.00%	71.00%	71.00%	
Macro Avg	73.00%	73.00%	71.00%	480
Weighted Avg	74.00%	71.00%	71.00%	480

Logistic Regression Algorithm

	Precision	Recall	F1-Score	Support
< 6	72.09%	73.25%	72.67%	756
>= 6	76.39%	75.32%	75.85%	843
Accuracy	74.36%	74.36%	74.36%	
Macro Avg	74.24%	74.29%	74.26%	1599
Weighted Avg	74.36%	74.34%	74.34%	1599

k-NN Algorithm

	Precision	Recall	F1-Score	Support
< 6	80.68%	79.70%	78.38%	735
>= 6	82.52%	83.39%	82.95%	864
Accuracy	81.68%	81.68%	81.68%	
Macro Avg	81.60%	81.55%	80.67%	1599
Weighted Avg	81.67%	81.69%	80.85%	1599

Conclusion and Recommendations

Binary classification of wines outperformed multi-classification by a significant amount, so if there is no need to know exact quality scores, it is better to consider quality scores based on a particular threshold. In both cases, the Random Forest algorithm would be the better choice for either scenario.

For future studies regarding multi-classification methods, it would be wise to consider the removal of outliers for that particular algorithm. The three models used are considered to be robust to outliers. However, as discussed in the Random Forest model, while outliers may not have an overall effect on general performance, there are some differences in the misclassifications of wine quality scores which may be important when considering the goal of the algorithm.

"I would base outlier removal on whether having incorrect classifications close to the top scores or the bottom scores matter with regards to 4 and 8. If one would prefer incorrectly classified score 4s as score 5, remove outliers, and vice versa. If one would prefer incorrectly classified score 8s as score 7, keep outliers. I don't think outlier removal matters at all for score 3, and if one wants any correctly scored 8, don't remove outliers."

As discussed in the multiclassification analysis above, exact misclassification scores would also determine which algorithm to use. The parameter selection for Random Forest could also be tuned to better handle misclassification of quality scores.

For binary classification, outlier removal may also make a significant difference depending on the model used. It has been shown that adjusting the threshold should make a difference in the algorithm's performance. In one paper, setting the threshold at 7 to detect 'excellent' quality wines greatly increased all metrics.

Adjusting the threshold for binary classification in an imbalanced dataset can impact accuracy for several reasons:

- Imbalance in Class Distribution:
    - In imbalanced datasets, accuracy might be high even if the classifier is just predicting the majority class all the time. For example, if 95% of the samples belong to class A, a classifier could achieve 95% accuracy by always predicting class A.
- Threshold Adjustment and Trade-off:
    - When you adjust the threshold for classification, you are essentially changing the trade-off between sensitivity (true positive rate) and specificity (true negative rate).
    - In imbalanced datasets, adjusting the threshold can be particularly useful. By lowering the threshold, you may increase the number of true positives (correctly identified minority class instances) at the expense of potentially more false positives (misclassifying majority class instances). This can improve sensitivity and, in some cases, overall accuracy.

Khilari, N., Hadawale, P., Shaikh, H., & Kolase, S. (2021). "Analysis of Machine Learning Algorithm to predict Wine Quality." International Research Journal of Engineering and Technology (IRJET), 8(12), 1082. https://www.irjet.net/ ISSN: 2395-0056, p-ISSN: 2395-0072.

- https://www.irjet.net/archives/V8/i12/IRJET-V8I12183.pdf