George Mason University
AIT 736
Summer 2023

Predicting Wine Quality Using KNN and ANN

Mitchell Breeden
An Major
Dedelolia Olungwe
Gregory Tress

Abstract

Wine production and consumption continues to be a highly profitable industry while wine quality continues to be dependent on individual human taste. The goal of this study is to determine wine quality using physiochemical properties of wine and wine tasters' ratings. The dataset used represents red and white wines sample of Vinho Verde, a region in Northern Portugal. Many classification techniques have been developed using the same dataset, however this study will have utilized K-nearest neighbors (KNN) and artificial neural networks (ANN) using TensorFlow to create an optimal model for predicting wine quality. This paper will explore the methods and algorithms as well as the accuracy results of using such models. These models will provide a tool for standardizing wine quality within the industry.

Table of Contents

Table of Figures

Introduction

Background and Rationale

Being the largest consumer of wine, the United States consumes a rate of 4.3 billion bottles per year (Alcohol.org, 2023). Wine has been consumed by people all over the world for thousands of years. Due to the existence of wine in many countries and cultures, it is difficult to pinpoint the exact origins of wine. Some evidence has suggested that "the earliest wine production came from Armenia, Georgia, and Iran, dating from 8000 to 5000 BC (Cooking, n.d.). Others have suggested that it was discovered in the Henan province of China dating back to 7000 BC (Hagan, 2020). But what is considered good wine? This project will explore the aspects of wine quality and develop a model to predict wine qualities.

Research

Wine consumption and wine collection have been a pastime for many all over the world. Currently, the most expensive bottle of wine on the market is a 1945 Domaine de la Romanee-Conti, priced at $558,000 (Kimball, 2023). The next highest acquired wine in history was the Screaming Eagle's Napa Valley Cabernet Sauvignon Vintage, priced at $500,000. The Screaming Eagle was considered a highly prized wine because it was acclaimed by many wine critics including Robert Parker, who gave the wine a score of 99 out of 100 (The CEO Magazine, 2022).

Wine ratings, originally developed by UC Davis considered the quality of wine based on key metrics including clarity, flavor and stability (Wallace, n.d.). Later in the 1980s, Robert Parker, a well-known wine critic further enhanced the wine rating to a 100-point system (Puckette, n.d.). Wine ratings are not based on the enjoyment of the wine, but rather the correctness of the wine according to industry standards of balance, length, complexity, and

intensity (Wallace, n.d.). However, industry standards can still be a subjective process and can be dependent on those critics who hold the highest reputation. This leaves wine connoisseurs with continuing to wonder what is considered quality wine.

Due to the time constraints of this project, the team was not able to develop and create their own data collection efforts. A review of existing quality wine datasets was conducted. Two wine quality datasets were open source and available for public use. These wine quality datasets were the Wine Quality from UCI Machine Learning Repository (Cortez, Cerdeira, Almeida, Matos, & Reis, 2009) and a Spanish Wine Quality dataset obtained from Kaggle.com (Fedesoriano, n.d.). The wine quality dataset from UCI Machine Learning Repository was selected for this project because features include several physiochemical properties, e.g., pH, residual sugar and acidity, while the Spanish Wine Quality dataset did not.

Motivation

The goal of this project is to develop predictive models using existing historical wine quality data to create a standard for determining quality wine. This can be utilized by wine producers, wine critics and consumers to create an industry standard.

Proposed Plan

Problem

The wine business is a very large and lucrative industry worldwide. However, there is no objective standard in considering what quality wine is. While many of the historical wine ratings have commonalities of key metrics such as balance, complexity, and flavor, it continues to be dependent on the critic providing the rating.

Proposed Solution

Using existing datasets on wine quality based on several physiochemical variables and sensory data, this project will develop a machine learning model to predict quality wines. The sample dataset used will be the Wine Quality Dataset (Cortez, Cerdeira, Almeida, Matos, & Reis, 2009). In analyzing both red and white wine samples, the predictive model will be based on the quality rating assigned to each wine sample considering 11 factors such as fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, and alcohol. The predictive model in this project is expected to provide guidance and quality metrics to vineyards and wine suppliers to improve and enhance their wines.

Dataset

The dataset provided by Cortez et al is structured in two CSV data files, representing red and white variants of the Portuguese "Vinho Verde" wine. There is no data about grape types, wine brand, or wine selling price. Instead, the input data is focused on the chemical properties of the wine. There are 1,599 rows of red wine data and 4,898 rows of white wine data. The output value for each row is a quality rating which is nominally scored on a scale from 0 to 10, with 10 being the highest quality. The quality rating is always a whole number, calculated from the median of at least 3 evaluations made by wine experts. In these datasets, the actual quality ratings for red wine range from 3 to 8 (inclusive), while the quality ratings for white wine range from 3 to 9 (inclusive).

Field Descriptions

In analyzing both red and white samples, the predictive model will be based on the quality rating assigned to each wine sampling considering the 11 factors in Table 1.

Input Variables	***Units***	**Description**
Fixed acidity	*g (tartaric acid)/L*	Physiochemical tests
Volatile acidity	*g (acetic acid)/L*	
Citric acid	*g/L*	
Residual sugar		
Chlorides	*g (sodium chloride)/L*	
Free sulfur dioxide	*mg/L*	
Total sulfur dioxide		
Density	*g/mL*	
pH		
Sulphates	*g (potassium sulphate)/L*	
Alcohol *(% vol.)*	*% vol.*	

Table 1*. For the purpose of quality assurance, physiochemical properties were investigated to reveal correlation.*

Output Variable	**Description**
Quality	Sensory data, score between 0 and 10

Table 2. *The quality of wine is scored from 0 to 10 – 0 (least quality), 10 (highest quality).*

Data Quality Assessment

Taking an initial look at the data, we observed that no entries were missing, and the CSV files were properly structured with no extraneous fields. There were no problems reading or parsing the files for analysis. There is no inherent ordering to the rows. All columns are numeric, non-negative measurements, with some columns having small decimal values (between 0 and 1) and other columns having larger ranges (1 to ~500); this suggested that normalization of the data may be useful for the analysis. The structure of the red and white wine data files is the same, suggesting that the same parsing and analysis methods may be applied to both files.

An analysis of correlation among features within the dataset was conducted. On the correlation plot on the red wine sample, there does not show any significant correlations (.80 or higher). There is an indication that there is a positive correlation between fixed acidity and density of .67. Looking at Figure 1, there also shows a correlation between free sulfur dioxide

and total sulfur dioxide, however it is likely the result of the two features representing the same type of physiochemical property.

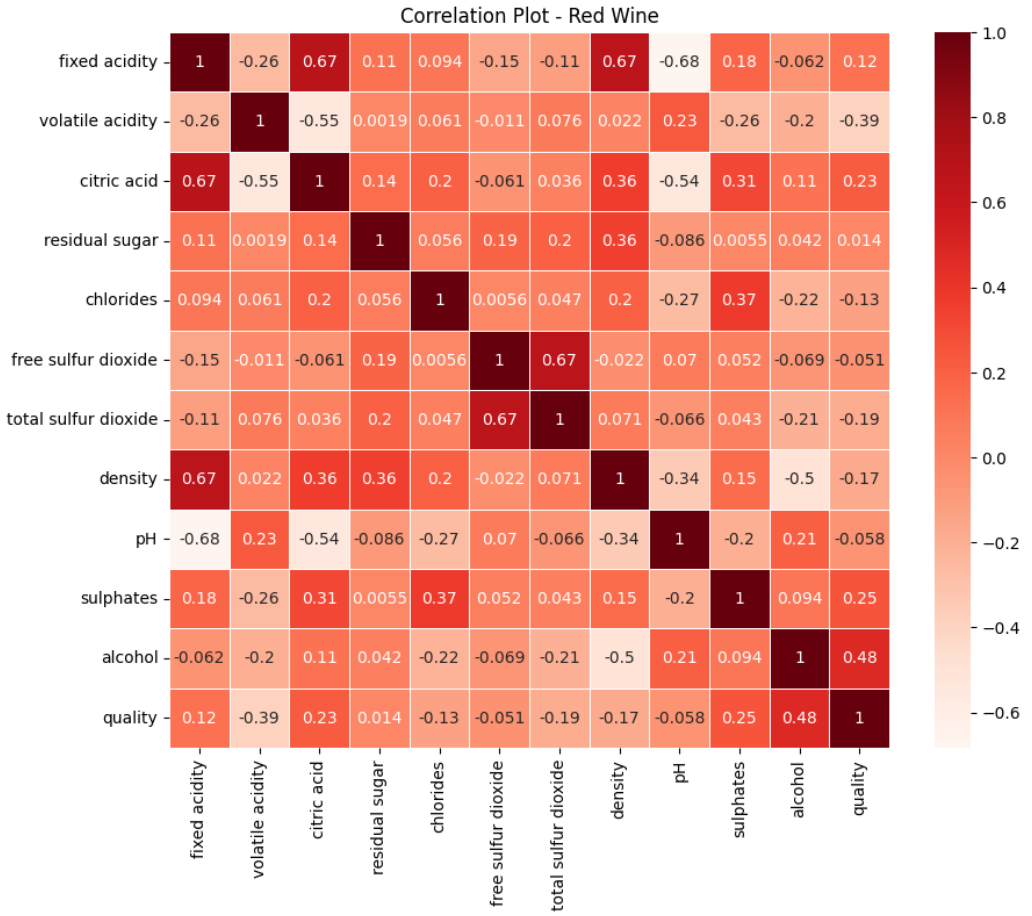

Figure 1: Red Wine Correlation Plot developed using Python

The correlation plot of the white wine sample shows slightly different results. This is a strong correlation of .84 between two features residual sugar and density. This indicates that as residual sugar increases, density will likely increase as well. All other features do not show a correlation.

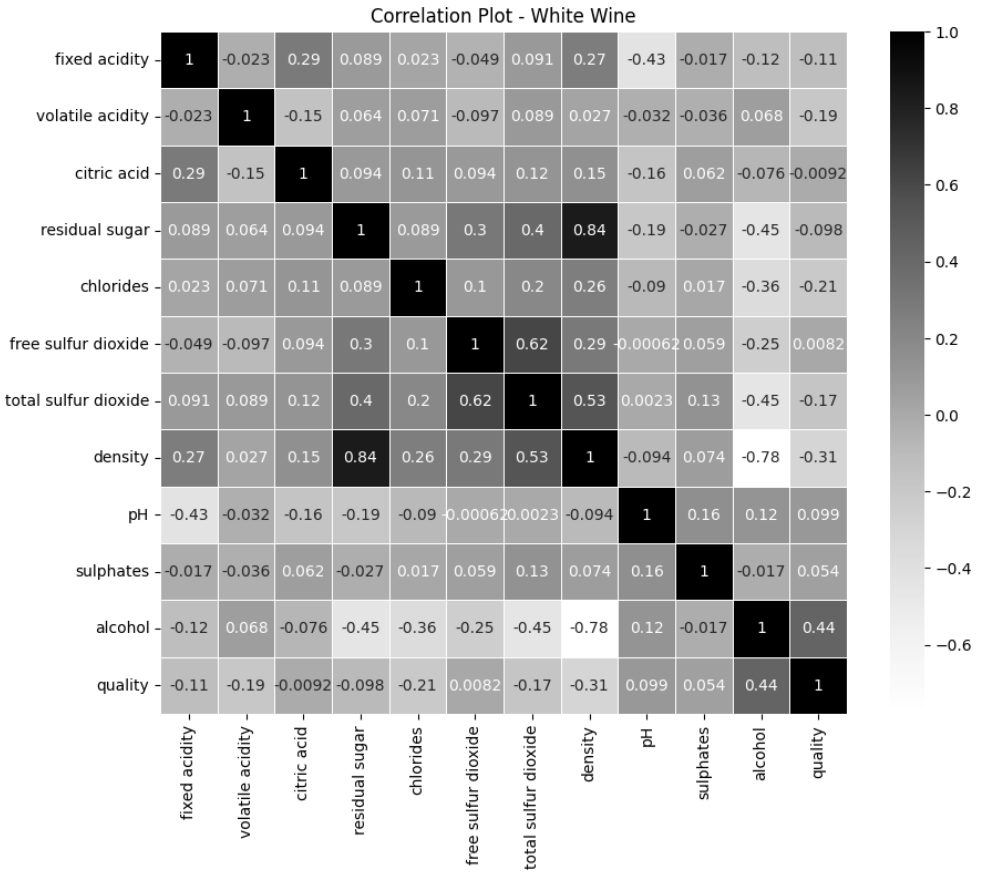

Figure 2: White Wine Correlation Plot developed using Python

The distribution of quality (output) values appears to follow a Gaussian pattern with many values in the “average” quality range, and a few examples rated very high or very low. (See figure.)

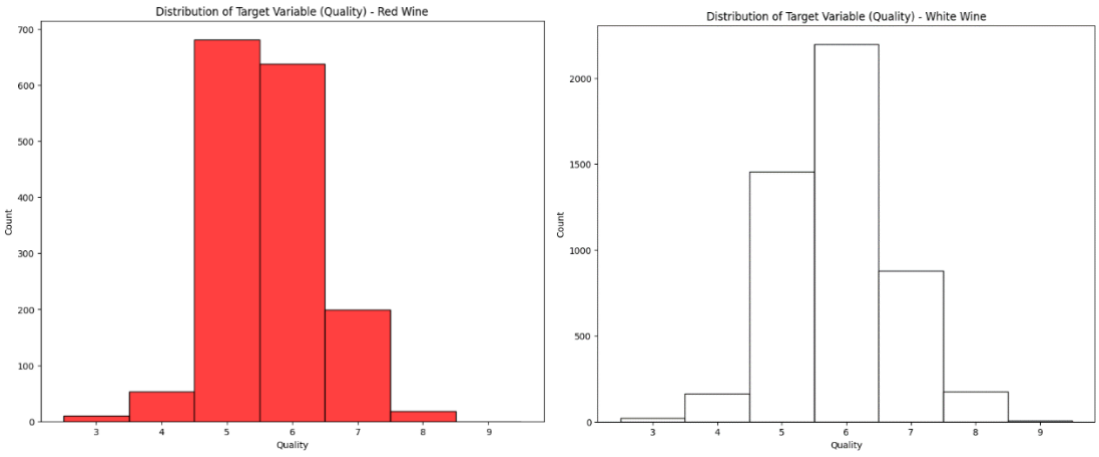

Figure 3: Quality distribution developed using Python

Because the quality (output) is always a whole number, there are a few possible approaches to the model:

- Apply regression to predict the numerical output value, based on the input features. The predicted output could optionally be rounded to a whole number to match the dataset format.
- Apply classification to predict the output value as a whole number. Each possible whole number value for which we have training examples is treated as a label (for example, each number from 3 to 9).
- Apply binary classification to predict whether the quality will exceed a chosen threshold. For reference, regarding the white wine dataset, there are 180 examples of quality 8 or greater (3.675% of examples), and 1,060 examples of quality 7 or greater (21.64% of examples). Meanwhile, regarding the red wine dataset, there are 18 examples of quality 8 or greater (1.12% of examples), and 217 examples of quality 7 or greater (13.57% of examples).

For data normalization, we plotted and observed the input features before and after normalization using a Yeo-Johnson power transformation. This transformation allows the models to be trained more efficiently.

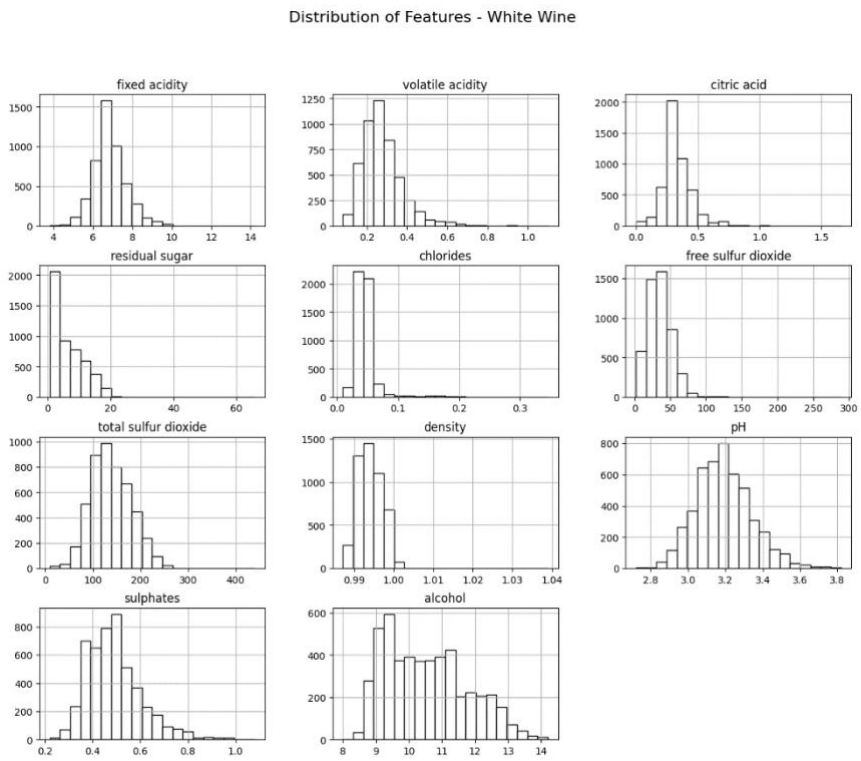

Figure 4: Distribution of features in white wine, developed using Python

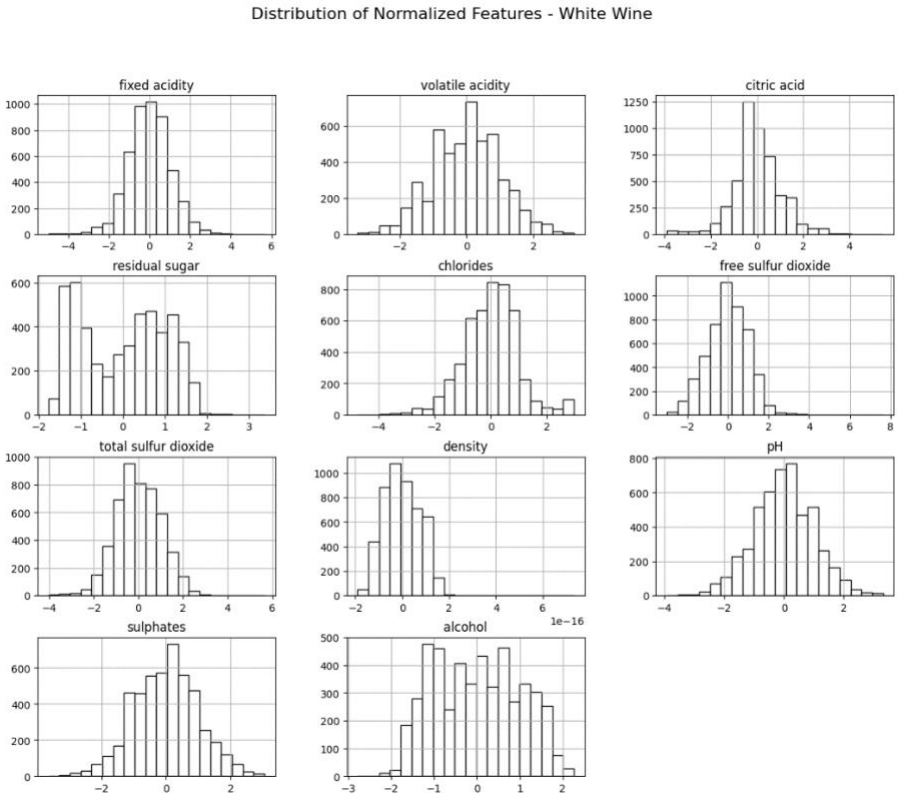

Figure 5: Distribution of white wine features after normalization, developed using Python

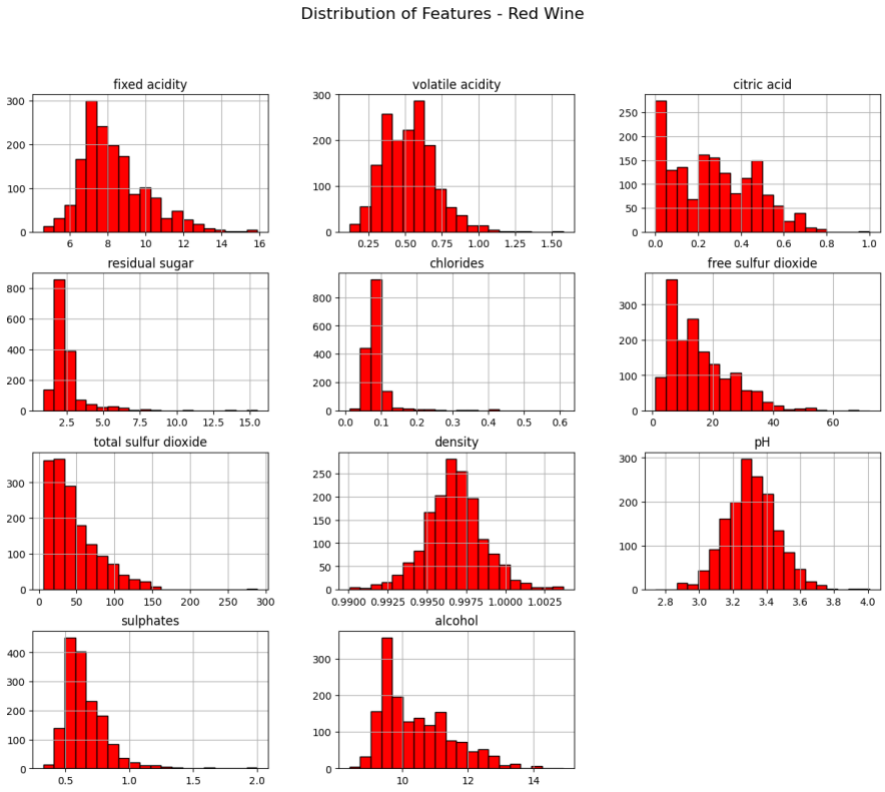

Figure 6: Distribution of features in red wine, developed using Python

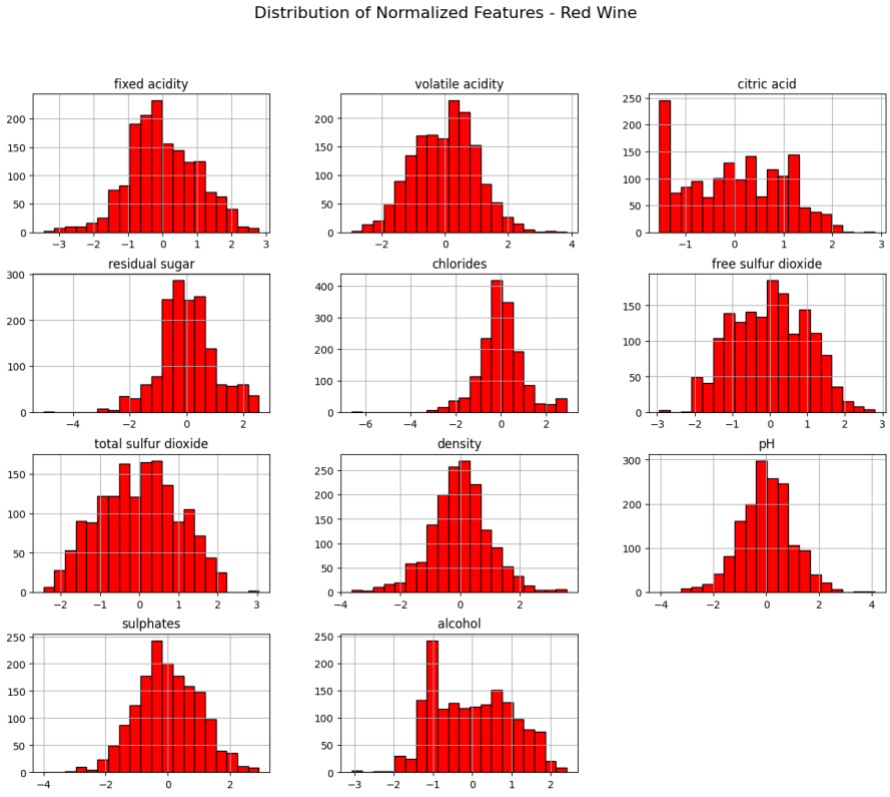

Figure 7: Distribution of red wine features after normalization, developed using Python

Analytics and Algorithms

KNN Classification

The first model that we wanted to use to classify the wine qualities was K-nearest neighbors (KNN). KNN is useful for classification tasks because of its ability to detect patterns within data which is exactly what we are trying to achieve with our dataset.

KNN: Methods

We started by splitting the data into an 80/20 train-test split for both the red and white wine datasets as this is the standard ratio to ensure enough training and test data for accurate results. After doing this, we created a hyperparameter grid containing 1, 3, 5, 7, 9, 11, 13, and 15 neighbors. For weights, we chose uniform (each of the k-nearest neighbors has an equal contribution in the prediction) and distance (the influence of each neighbor on the prediction is proportional to its distance from the new instance). For metrics, we chose Euclidean (calculates the straight-line distance between two points in Euclidean space), Manhattan (measures the distance between two points as the sum of the absolute differences between their coordinates), Minkowski (a generalized form that includes both Euclidean and Manhattan distances as special cases) and Chebyshev (calculates the maximum absolute difference between the coordinates of two points).

KNN: Algorithm

To run the algorithm, we utilized the KneighborsClassifier class from scikit-learn. We then completed a grid search with our hyperparameters. This allowed us to run every combination of our hyperparameters in order to detect which combination provided the highest accuracy. This process consisted of 5 folds for each of 64 candidates and resulted in 320 fits. From that, we were able to determine that the optimal number of neighbors was 15, the optimal

weight was distance, and the optimal metrics were Euclidean for Red Wine and Manhattan for White wine. We then re-ran the algorithm with those parameters and calculated accuracy scores using the accuracy_score function from scikit-learn.

KNN: Results

We were able to obtain accuracy scores of 65.94% for red wine and 67.76% for white wine. Next, we wanted to see if we could gather more insights using an artificial neural network with TensorFlow.

ANN with Tensorflow

The next model uses an artificial neural network (ANN) technique using TensorFlow. This model utilized Keras sequential model and TensorBoard to visualize results with the Python programming environment. The technique will be taking different approaches of regression, classification, and binary classification.

ANN: Regression model

Using the regression approach attempts to predict a real number output, as close as possible to the label. In order to determine the loss function between the predicted number and the real number, Mean Squared Error (MSE) was used. This computes the mean of squares of errors between labels and predictions. The optimization uses the Adam algorithm for gradient descent (Kingma, 2014). The Adam optimizer has a built-in default learning rate of 0.001, which is sufficient for our testing. The loss function and optimizer have other tuning options available, which are outside the scope of our experiments.

A review of input data was conducted to determine if normalization should be applied. Ideally, we want to see the result converge quickly and consistently using the minimum number

of epochs. The following code snippet is an example of how the model is initially configured with a loss function and optimizer.

```
model.compile(loss = tf.keras.losses.MeanSquaredError(),
              optimizer = tf.keras.optimizers.Adam())
```

After the model is compiled, the model is fit to the data as shown in the following code snippet.

```
model.fit(features, labels,
          epochs=100, verbose=1,
          callbacks=[tensorboard_callback])
```

The number of epochs can be specified, which increases accuracy (in other words, reduces loss) with greater epochs, but takes longer to execute. In our experiments, the model labels are the whole numbers of wine quality as provided in the dataset, and the features consist of all 11 attributes of wine measurements in the dataset.

ANN: Averaging results

Due to the random state when creating the model, the behavior will be different for different executions of the same code. An example of this is shown in Figure 8 which shows the behavior of the loss function for different executions of the red wine dataset, using non-normalized input data, and a simple model with one layer. Each model is created with identical data, but due to the random state, takes a different amount of time to converge near the best fit.

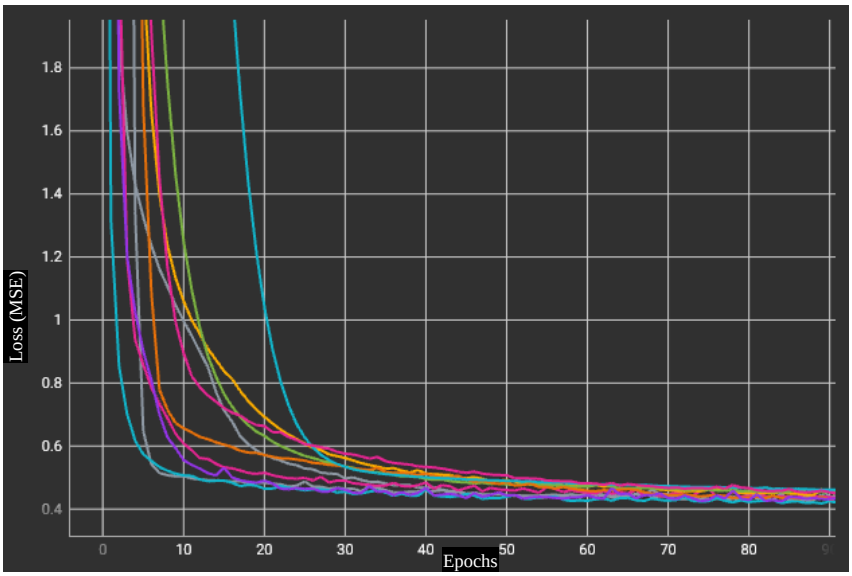

Figure 8: Loss function for 10 executions of red dataset, non-normalized, developed using Python

Due to this random behavior, our results will show an average of multiple executions for each approach. This is implemented in code using a loop that iteratively creates a model and saves the result to a log file. The log files are aggregated together, and the average result is displayed using TensorBoard. In our experiments, we adjusted the loop between 4 and 10 executions, depending on the complexity of the model. Due to time constraints, it was not feasible to increase the loop to a much larger number of executions. Using around 10 executions is sufficient to show the end behavior of the model. Figure 9 shows the averaged results of the loss function for both the red and white datasets. The subsequent graphs in this report will show these averaged results.

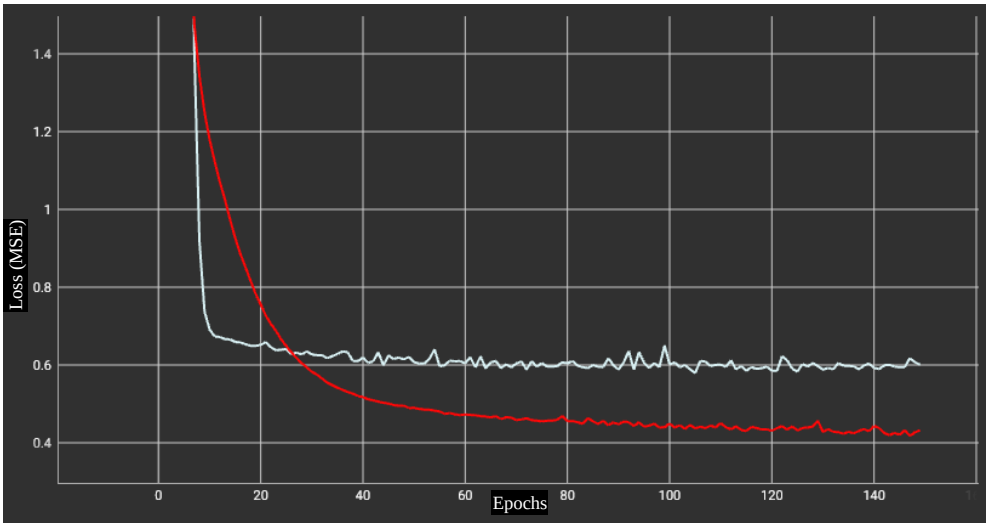

Figure 9: Loss function for red and white datasets, non-normalized, developed using Python

The loss function for white wine converges to about 0.60 and for red wine converges to about 0.42. There is little change in the loss function after about 100 epochs, and we get diminishing returns in terms of increased accuracy after that point.

ANN: Impact of normalization

The dataset attributes are then normalized using a Z-score method, which transforms the data to target a mean of 0 and a typical deviation of 1 (Rustagi). This can be done by subtracting the mean from each attribute value and then dividing it by the standard deviation for that attribute. Note that only the input attributes are normalized with this method; the output labels are not changed. Now, using normalized inputs, the results have greater overall accuracy and show a smoother curve; but take longer to converge near the start of execution. This additional time is likely due to some normalized attributes having less impact on the output when the weights are changed, compared to the non-normalized attributes having larger numeric values.

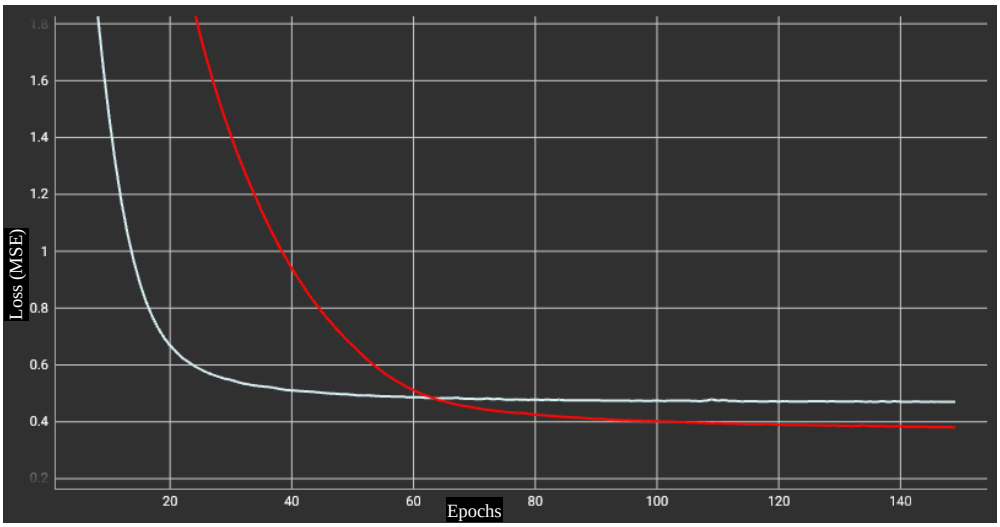

Figure 10: Loss function for red and white dataset, normalized inputs, developed using Python

Using normalized attributes, the end behavior of the loss function is reduced further, compared to the non-normalized inputs. Now, the loss for red wine is reduced to 0.40 after about 100 epochs and continues to decrease slightly to 0.38 after about 150 epochs. In contrast, the white wine dataset has about 3 times the number of rows as the red wine data set. The white wine model similarly shows improvement compared to the non-normalized inputs, from 0.60 after 100 epochs previously, now reduced to 0.48 after 100 epochs. The normalized inputs will be used for the remainder of our experiments.

ANN: Regression Layers

To build the model used for regression, a sequential model with dense layers is created. This means that the model will have a series of layers connected in a linear sequence, with the neurons in each layer connected to all neurons in the layer immediately preceding it. The first layer has an equal number of neurons to the number of features in the dataset. The activation function used for most neurons in the model is the Rectified Linear Unit (ReLU) which will output either a zero (if the input is negative) or positive value (and equal to the input, if the input

is positive) (Layer activation functions, n.d.). The final layer is a single node which outputs the predicted result. Creating the described layer structure can be done with a code snippet like this:

```
model = tf.keras.Sequential([
   layers.Dense(11, activation="relu"),
   layers.Dense(1)
 ])
```

The model is then compiled, optimized, and fitted to the data. For our experiments, the same model structure was created separately for both the red wine and white wine datasets. In each case, the full dataset is used to train the model. In evaluation of the model, we observe the loss function to show the overall error in predicting the result.

The goal in training the model is to achieve a loss function close to zero, but also to operate in a reasonable timeframe and limited complexity. With some trial and error, we can adjust the layers of the model, and the number of epochs used for training, until we obtain a model that has sufficient accuracy. The table below shows the time impact of increasing the number of epochs for different sizes of dataset. In general, the time increases approximately linearly with the size of the dataset and with the number of epochs.

Size of Dataset (white/red factor)	**Train 100 epochs, 1 layer**	**Train 200 epochs, 1 layer**	**2x epochs time factor**
White dataset (4898) (w/r: 3.06 x)	30.5 s (w/r: 3.2 x)	53.6 s (w/r: 2.9 x)	1.76 x
Red dataset (1599)	9.5 s	18.3 s	1.93 x

Figure 11: Performance impact of increased epochs

After several experiments, it was noted that there was not much improvement in accuracy after approximately 150 epochs, so further analysis was not needed. In general, we can observe the loss curve to see where it “flattens out”; that is, the accuracy does not significantly improve

by adding more epochs after that point. This means we can limit the number of epochs and limit the time required to execute the experiment without reducing accuracy.

The next approach to optimize the training model was to determine if changing of the structure of the layers was needed. For example, we can add a second layer of 6 neurons using this code:

```
model = tf.keras.Sequential([
    layers.Dense(11, activation="relu"),
    layers.Dense(6, activation="relu"),
    layers.Dense(1)
  ])
```

The result indicated a less than 1% improvement in the model accuracy when a second layer was added. This was not a significant improvement, so we continued to change the layers to find a better solution. Adding larger layers (more neurons per layer) and more total layers did generally show improvement in accuracy (that is, reduce loss). For example, updating the red wine model to include 5 layers with up to 99 neurons in some layers did reduce the loss to under 0.11, which is a significant improvement from the 1 layer model shown earlier with a loss of 0.38.

ANN: Classification

Based on the regression model, there is a good understanding of how to structure the classification model to obtain better accuracy. For classification, we will use a sequential model with dense layers, similar to regression. The number of neurons in the output layer matches the number of possible labels that we are predicting.

The Keras models are designed to provide a classification of N items by outputting a whole number value from 0 to N-1. Therefore, we must normalize our labels to be zero-indexed. All the quality labels ranged from 3 to 9 (red 3 to 8, white 3 to 9). Subtracting 3 from all labels,

we obtain a range of 0 to 5 or 0 to 6 respectively. This results in 6 possible labels for red and 7 possible labels for white. Constructing the model just described can be done with the following code:

```
model = tf.keras.Sequential([
   layers.Dense(11, activation="relu"),
   layers.Dense(7, activation="softmax")
  ])
```

Differing from the regression approach, Softmax activation will be used as the activation function for classification. This approach compares the probability for each output class. When compiling, optimizing, and fitting this model, SparseCategoricalCrossentropy is used to determine loss function and Adam for the optimizer. SparseCategoricalCrossentropy is a loss function provided by Keras for use with two or more integer label values (Probabilistic losses, n.d.). The accuracy metric tells us the rate of successful classification. Compiling the model can be done as follows:

```
model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(),
              metrics=['accuracy'])
```

The figures below show the loss function (left) and the accuracy metric (right) from a basic model with one layer. Training was done for 100 epochs. The accuracy percentage is noted to plateau at 62% for red and 57% for white.

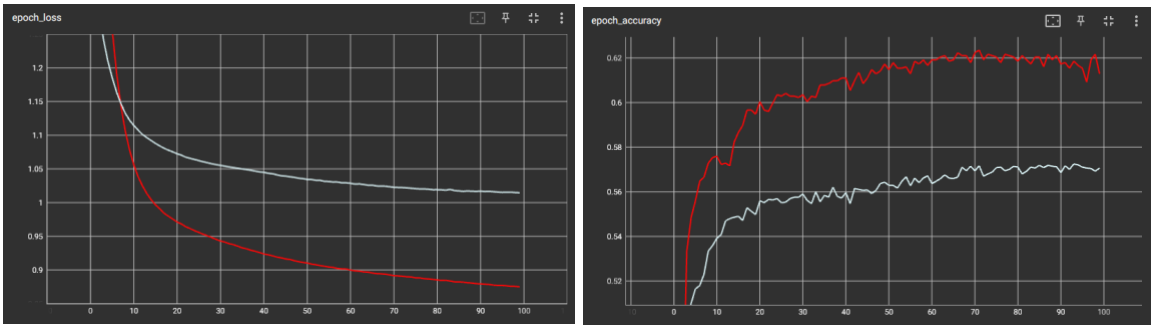

Figure 12: Epoch loss and epoch accuracy in classification (one layer and 100 epochs), developed using Python

Overal several iterations of the experiment, we increased the layers in the model and observed an increase in accuracy. Using 3 intermediate layers of 99 neurons each, both the red and white models obtained accuracy over 95%. Using 5 intermediate layers of 99 neurons each resulted in over 99% accuracy for both models after 150 epochs. The figures below show the loss function decreasing while the epcoch accuracy increases. There is some jagged behavior in the graph where the accuracy drops by 3-4% in a short period, then corrects itself after a few more epochs. This may be a result of the learning rate where the model overfits and then corrects itself.

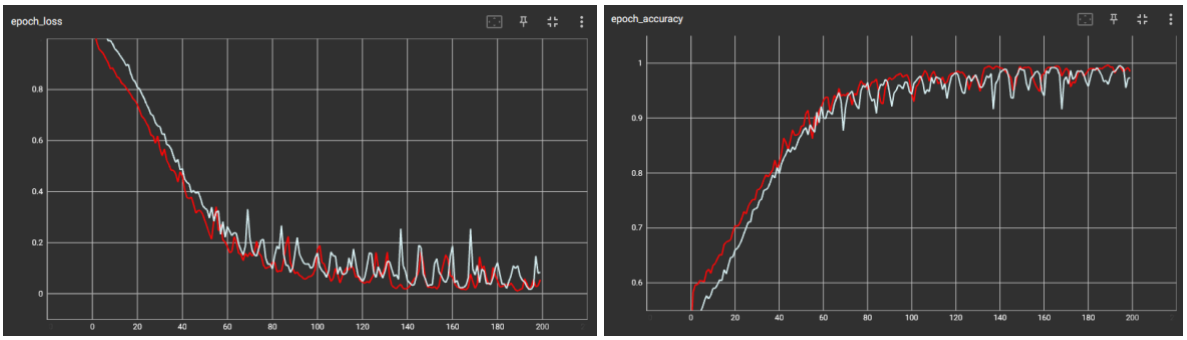

Figure 13: Epoch loss and accuracy in classification (5 layers and 150 epochs), developed using Python

Again, we attempted a cross-dataset evaluation to determine if the model from one dataset may be used to accurately classify data from the other dataset. In both cases (red model with white data, and vice versa), the resulting accuracy of 41% is too low to accurately classify the data. The conclusion is that the quality scales and subjective taste factors between red and white wine are different enough that the measurable chemical properties of the wine alone are not sufficient to make a prediction about a quality rating.

ANN: Binary classification

The last approach is binary classification, which can be used to determine if the wine meets the criteria for high quality wine. To construct this experiment, we establish categories of "excellent" wine with a quality score of 8 or greater, and "good" wine with a quality score of 7

or better. The output of the model will be a binary value of either 0 (indicating that the wine does not meet the threshold quality in the given category) or 1 (indicating that it does meet or exceed the threshold). Similarly, the training labels are normalized to 0 or 1 based on this definition. A binary model can be created as follows:

```
model = tf.keras.Sequential([
   layers.Dense(11, activation="relu"),
   layers.Dense(1, activation="sigmoid")
 ])
```

Here we use a sigmoid activation function, which returns a value between 0 and 1.

For binary classification, we use a similar strategy as in labeled classification to adjust the layer structure to improve accuracy. First, we consider the "excellent" wine category of quality 8 and above. Less than 4% of white wine and less than 2% of red wine data meets this quality threshold. Using a model with 2 intermediate layers of 99 neurons each, this approach showed great accuracy of 99% percent. Additionally, when applying the model to the other dataset, it showed an accuracy of 95% when classifying "excellent" wine.

Accuracy	White model	Red model
White dataset	0.9588	0.7607
Red dataset	0.7730	0.9987

Accuracy	White model	Red model
White dataset	0.9959	0.9500
Red dataset	0.9656	0.9981

Next, we use a similar approach to classify "good" wine with quality 7 and above. The accuracy was also high at 95% to 99%; however, generalizing to the other dataset did not yield the same results as the "excellent" classification.

ANN: Results

The overall results from the neural network approaches showed that we can consistently predict wine quality classification based on set criteria of "excellent" and "good" wine with high accuracy (close to 99% depending on complexity). Accurate classification requires multiple layers and a large number of neurons. As expected, the more complex models with more layers, more neurons per layer, and more epochs of execution did perform better, but all models eventually encounter diminishing returns that do not justify further increasing the complexity. Normalization of the input data is recommended for consistency and accuracy.

In addition, we can also evaluate one model using the other dataset (red model using white wine data, and vice versa). Our hypothesis was that the loss would be similar when evaluating cross-dataset, based on the idea that the measured chemical properties of the wine would correlate to quality, regardless of the wine color. However, the actual results indicated that the model created with one dataset did not have good accuracy in predicting results of the other dataset. The MSE was approximately doubled when evaluating cross-dataset. This could be the result of several factors. The datasets provided are limited in number of features, and other chemical properties of the wine that affect the taste are likely not captured in the given measurements. As wine quality is subjective to human tastes and preferences, it may be difficult to find scientific measurements that capture universal tastes. For example, some people may experience a better taste correlated with higher sulphates in red wine, but not in white wine.

Future work, beyond the scope of our experiments, may examine whether the model can be simplified and still maintain high accuracy. This may include: eliminating some features that do not significantly affect the result, changing the tuning parameters in the activation functions and optimizer, and adding non-sequential layers with additional transformations.

Summary

This study explores machine learning's application in wine quality assessment, using methodologies for valuable insights. The exploratory data analysis (EDA) phase examined correlations and visualized feature distributions to gain insights into the dataset's structure. The next step involves implementing the K-Nearest Neighbors (KNN) algorithm, which produces predictive results. The red wine dataset achieves 65.94%, and white wine attains 67.76% accuracy, highlighting the algorithm's proficiency in recognizing data patterns. By employing GridSearch CV, the KNN hyperparameters were optimized, yielding a model that accurately predicts wine quality. In addition to KNN, an Artificial Neural Network (ANN) implemented through TensorFlow was used to gain deeper insights into the datasets. TensorFlow delivers robust predictions, with accuracy surpassing 99% in identifying excellent quality wines. Strategic enhancements of network layers lead to further predictive improvements.

The research establishes a foundational understanding of systematic wine quality assessment. Its implications are substantial, potentially advancing wine classification and fostering objective quality evaluation. These insights resonate within the industry and with enthusiasts, bridging machine learning and enology in an engaging interdisciplinary venture.

Recommendations

It would be important to consider weather, climate, and farming practices as they substantially influence the chemistry of grapes, thereby profoundly impacting the resultant wine's characteristics. The interplay of temperature, sunlight, rainfall, and soil composition plays a pivotal role in shaping attributes like acidity, sugar content, and flavor compounds within the grapes (Sun, 2023).

Warm temperatures and abundant sunlight enhance sweetness and alcohol levels, while cooler climates contribute to heightened acidity, lending wines a refreshing and crisp quality. Seasonal predictions based on weather trends empower vineyard managers to optimize harvest timings, ensuring desired sugar levels and acidity. Additionally, controlled viticultural practices, encompassing canopy management, irrigation, and soil analysis, enable the modulation of grape ripening, thus providing the opportunity to fine-tune attributes for specific flavor profiles.

By pairing the model results with established principles of wine taste, particularly concerning attributes like acidity, the alignment between predictive outcomes and anticipated outcomes comes to light. The analysis of feature correlations reveals insights into how acidity, among other factors, contributes to wine quality. As acidity levels influence the overall balance and freshness of wines, it is heartening to observe that the models successfully capture these nuances.

References

Alcohol.org. (2023, 01 17). *Worldwide Production & Consumption Rates of Beer and Wine*. Retrieved from American Addiction Centers Alcohol.org: https://alcohol.org/guides/beer-wine-production-consumption-worldwide/

Cooking, S. o. (n.d.). *Wine*. Retrieved from Science of Cooking: https://www.scienceofcooking.com/food-and-wine/history-of-wine.html

Cortez, P., Cerdeira, A., Almeida, F., Matos, T., & Reis, J. (2009). *Wine Quality*. Retrieved from UCI Machine Learning Respository: https://doi.org/10.24432/C56S3T.

Fedesoriano. (n.d.). *Spanish Wine Quality Dataset*. Retrieved from Kaggle: https://www.kaggle.com/datasets/fedesoriano/spanish-wine-quality-dataset

Hagan, M. (2020, 10 13). *Wine History: Exploring Wine's History and Origins*. Retrieved from Usual Wines: https://usualwines.com/blogs/knowledge-base/history-of-wine

Kimball, S. (2023). *Top 5 Most Expensive Bottles of Wine Ever Sold*. Retrieved from Crush Wine XP: https://crushwinexp.com/top-5-expensive-bottles-wine-ever-sold/

Kingma, D. P. (2014). *Adam: A Method for Stochastic Optimization.* Retrieved from https://arxiv.org/abs/1412.6980

Layer activation functions. (n.d.). Retrieved from Keras.io: https://keras.io/api/layers/activations/

Probabilistic losses. (n.d.). Retrieved from Keras.io: https://keras.io/api/losses/probabilistic_losses/

Puckette, M. (n.d.). *A Pragmatic Approach to Using Wine Ratings*. Retrieved from Wine Folly: https://winefolly.com/tips/wine-ratings-explained/

Rustagi, D. (n.d.). *Normalize A Column In Pandas.* Retrieved from https://www.geeksforgeeks.org/normalize-a-column-in-pandas/

Sun, e. a. (2023). *Viticultural Manipulation and New Technologies to Address Environmental Challenges Caused by Climate Change*. Retrieved from https://doi.org/10.3390/cli11040083

The CEO Magazine. (2022, 07 01). *The World's Most Expensive Wines Are a Testimony of Survival*. Retrieved from CEO Magazine: https://www.theceomagazine.com/lifestyle/food-beverage/most-expensive-wines/

Wallace, K. (n.d.). *Wine Ratings 101*. Retrieved from Wine School: https://www.vinology.com/wine-ratings-101/

Appendix A: Exploratory Analysis and Preprocessing Code

https://colab.research.google.com/drive/1C1ezIXc2d_15aixjMwSCikMKqwZavePN?usp=sharing

```
# AIT-736 (Summer 2023)
# Group 1
#
# https://archive.ics.uci.edu/dataset/186/wine+quality
#
# Instructions:
#  - Upload files to root dir of this project runtime:
#    - winequality.names
#    - winequality-red.csv
#    - winequality-white.csv
```

Import Libraries

```
[ ]  import matplotlib.pyplot as plt
     import numpy as np
     import os
     import pandas as pd
     import seaborn as sns
     import tensorflow as tf
     import time
     from tensorflow.keras import layers
     from sklearn.neighbors import KNeighborsClassifier
     import matplotlib.pyplot as plt
     from scipy import stats
     from sklearn.preprocessing import PowerTransformer
     from sklearn.preprocessing import StandardScaler
     from sklearn.model_selection import train_test_split
     from sklearn.neighbors import KNeighborsClassifier
     from sklearn.metrics import accuracy_score, classification_report
     from sklearn.model_selection import GridSearchCV
     from imblearn.over_sampling import RandomOverSampler
     from imblearn.under_sampling import RandomUnderSampler
     from sklearn.datasets import make_classification
     from sklearn.model_selection import train_test_split
     from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import classification_report, accuracy_score
     from imblearn.over_sampling import RandomOverSampler
     from sklearn.model_selection import train_test_split
```

```
[ ]  # Optional: if data files need to be read from drive location
     from google.colab import drive
     drive.mount('/content/drive')
```

Exploratory Data Analysis

```
# Read the red and white wine CSV files into DataFrames
winequality_red_df = pd.read_csv("winequality-red.csv", delimiter=";")
winequality_white_df = pd.read_csv("winequality-white.csv", delimiter=";")

# Create a dictionary to hold the wine types and their corresponding DataFrames
wine_types = {
    'Red Wine': winequality_red_df,
    'White Wine': winequality_white_df
}

# Iterate through each wine type and perform exploratory analysis
for key, val in wine_types.items():
    # Determine color and colormap based on wine type
    color = 'Red' if key == 'Red Wine' else 'White'
    cmap = 'Reds' if key == 'Red Wine' else 'Greys'

    # Print the preview (first few rows) of the DataFrame
    print(f'{key} Preview:')
    print(val.head())

    # Print summary information about the DataFrame
    print(f'\n{key} Info:')
    print(val.info())

    # Create a correlation heatmap for the DataFrame
    plt.figure(figsize=(10, 8))
    sns.heatmap(val.corr(), annot=True, cmap=cmap, linewidths=0.5)
    plt.title(f'Correlation Plot - {key}')
    plt.show()

    # Plot the distribution of continuous features
    feature_distributions = val.drop('quality', axis=1).hist(figsize=(15, 12), bins=20, color=color, edgecolor='black')
    plt.suptitle(f'Distribution of Features - {key}', fontsize=16)
    plt.show()

    # Plot the distribution of the target variable (quality)
    plt.figure(figsize=(10, 8))
    sns.histplot(data=val['quality'], bins=[2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5], kde=False, color=color, edgecolor='black')
    plt.title(f'Distribution of Target Variable (Quality) - {key}')
    plt.xlabel('Quality')
    plt.ylabel('Count')
    plt.show()
```

Preprocessing

```
[ ]  # Function to normalize the features of a DataFrame
     def normalize(df, wine_type):
         """
         Normalize the features of a DataFrame using the Box-Cox transformation.

         Parameters:
         df (pandas.DataFrame): Input DataFrame containing features and target variable.
         wine_type (str): Type of wine ('Red Wine' or 'White Wine').

         Returns:
         X_normalized (numpy.ndarray): Normalized feature values.
         y (pandas.Series): Target variable values.
         """
         color = 'Red' if wine_type == 'Red Wine' else 'White'

         # Separating the features and target variable
         X = df.drop('quality', axis=1)
         y = df['quality']

         # Applying the Box-Cox transformation to normalize the skewed features
         power_transformer = PowerTransformer(method='yeo-johnson')
         X_normalized = power_transformer.fit_transform(X)

         # Converting to DataFrame to visualize the distributions after normalization
         X_normalized_df = pd.DataFrame(X_normalized, columns=X.columns)

         # Plotting the distribution of the normalized continuous features
         normalized_feature_distributions = X_normalized_df.hist(figsize=(15, 12), bins=20, color=color, edgecolor='black')
         plt.suptitle(f'Distribution of Normalized Features - {wine_type}', fontsize=16)
         plt.show()

         return X_normalized, y

     # Normalize features for red wine dataset
     X_red, y_red = normalize(winequality_red_df, 'Red Wine')

     # Normalize features for white wine dataset
     X_white, y_white = normalize(winequality_white_df, 'White Wine')
```

Appendix B: KNN Classification Code

KNN

```
# Function to perform k-Nearest Neighbors (KNN) classification
def knn(X, y):
    """
    Perform k-Nearest Neighbors (KNN) classification with hyperparameter tuning.

    Parameters:
    X (numpy.ndarray): Feature values.
    y (pandas.Series): Target variable values.

    Returns:
    best_accuracy_test (float): Best accuracy achieved on the testing set.
    """
    # Splitting the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    # Hyperparameter grid for KNN
    param_grid = {
        'n_neighbors': [1, 3, 5, 7, 9, 11, 13, 15],
        'weights': ['uniform', 'distance'],
        'metric': ['euclidean', 'manhattan', 'minkowski', 'chebyshev']
    }

    # Creating the grid search with 5-fold cross-validation
    grid_search = GridSearchCV(estimator=KNeighborsClassifier(), param_grid=param_grid,
                               cv=5, n_jobs=-1, verbose=1, scoring='accuracy')

    # Fitting the grid search to the data
    grid_search.fit(X_train, y_train)

    # Getting the best hyperparameters and the corresponding accuracy
    best_hyperparams = grid_search.best_params_

    print(best_hyperparams)

    # Creating the KNN model with the best hyperparameters
    best_knn_model = KNeighborsClassifier(**best_hyperparams)

    # Training the model
    best_knn_model.fit(X_train, y_train)

    # Predicting on the testing set
    y_pred_best = best_knn_model.predict(X_test)

    # Calculating the accuracy on the testing set
    best_accuracy_test = accuracy_score(y_test, y_pred_best)

    return best_accuracy_test

# Calculate KNN accuracy for red wine dataset
accuracy_red = knn(X_red, y_red)

# Calculate KNN accuracy for white wine dataset
accuracy_white = knn(X_white, y_white)

# Print the results
print(f'Red Wine Accuracy: {accuracy_red * 100:.2f}%')
print(f'White Wine Accuracy: {accuracy_white * 100:.2f}%')
```

Appendix C: ANN with TensorFlow Coding

TensorFlow

```
[ ] #
    # Tensorflow
    #

    import tensorboard
    from datetime import datetime
    %load_ext tensorboard

    # info
    print("TensorFlow version:", tf.__version__)
    print("TensorBoard version:", tensorboard.__version__)

    # Plots can be color coded uses these colors:
    # red
    # #ff0707
    # white
    # #d8f0f3

    # Clear any logs from previous runs
    !rm -rf ./logs/

    # Define the Keras TensorBoard callback.
    logdir="logs/fit/" + datetime.now().strftime("%Y%m%d-%H%M%S")
    tensorboard_callback_red = tf.keras.callbacks.TensorBoard(log_dir=logdir+"-red")
    tensorboard_callback_white = tf.keras.callbacks.TensorBoard(log_dir=logdir+"-white")

    # Extract relevant columns  (https://www.tensorflow.org/tutorials/load_data/csv)
    # "features" = everything except quality column
    # "labels" = quality column
    winequality_red_features = winequality_red_df.copy()
    winequality_red_labels = winequality_red_features.pop("quality")
    winequality_white_features = winequality_white_df.copy()
    winequality_white_labels = winequality_white_features.pop("quality")

    # Normalization
    # https://www.geeksforgeeks.org/normalize-a-column-in-pandas/#
    # z-score method (often called standardization) transforms the info into distribution with a mean of 0 and a typical deviation of 1
    print()
    for i in range(len(winequality_white_features.columns)):
      colname = winequality_white_features.columns[i]
      winequality_red_features[colname] = (winequality_red_features[colname] - winequality_red_features[colname].mean()) / winequality_red_features[colname].std()
      winequality_white_features[colname] = (winequality_white_features[colname] - winequality_white_features[colname].mean()) / winequality_white_features[colname].std()

    # Regression
    print("Regression")
    # Number of trials to perform
    trials = 4
    for trial in range(trials):
      print("== Start trial " + str(trial))
```

```
[ ]   # Red
      print("Building red model")
      t = time.perf_counter()
      model_red = tf.keras.Sequential([
        layers.Dense(11, activation="relu"),
        layers.Dense(99, activation="relu"),
        layers.Dense(99, activation="relu"),
        layers.Dense(99, activation="relu"),
        layers.Dense(33, activation="relu"),
        layers.Dense(1)
      ])

      model_red.compile(loss = tf.keras.losses.MeanSquaredError(),
                        optimizer = tf.keras.optimizers.Adam())
      model_red.fit(winequality_red_features, winequality_red_labels,
                    epochs=100, verbose=0,
                    callbacks=[tensorboard_callback_red])
      print("Elapsed: " + str(time.perf_counter() - t))

      # White
      print("Building white model")
      t = time.perf_counter()
      model_white = tf.keras.Sequential([
        layers.Dense(11, activation="relu"),
        layers.Dense(99, activation="relu"),
        layers.Dense(99, activation="relu"),
        layers.Dense(99, activation="relu"),
        layers.Dense(33, activation="relu"),
        layers.Dense(1)
      ])

      model_white.compile(loss = tf.keras.losses.MeanSquaredError(),
                          optimizer = tf.keras.optimizers.Adam())
      model_white.fit(winequality_white_features, winequality_white_labels,
                      epochs=100, verbose=0,
                      callbacks=[tensorboard_callback_white])
      print("Elapsed: " + str(time.perf_counter() - t))


    # Classification
    print("Classification")
    # Normalize to start with 0
    winequality_red_labels = winequality_red_labels - 3
    winequality_white_labels = winequality_white_labels - 3
    # Number of trials to perform
    trials = 4
    for trial in range(trials):
      print("== Start trial " + str(trial))
```

```
# Red
print("Building red model")
t = time.perf_counter()
model_red = tf.keras.Sequential([
  layers.Dense(11, activation="relu"),
  layers.Dense(99, activation="relu"),
  layers.Dense(99, activation="relu"),
  layers.Dense(99, activation="relu"),
  layers.Dense(99, activation="relu"),
  layers.Dense(99, activation="relu"),
  layers.Dense(7, activation="softmax")
])
model_red.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),
            optimizer=tf.keras.optimizers.Adam(),
            metrics=['accuracy'])
model_red.fit(winequality_red_features, winequality_red_labels,
              epochs=200, verbose=0,
              callbacks=[tensorboard_callback_red])
print("Elapsed: " + str(time.perf_counter() - t))

# White
print("Building white model")
t = time.perf_counter()
model_white = tf.keras.Sequential([
  layers.Dense(11, activation="relu"),
  layers.Dense(99, activation="relu"),
  layers.Dense(99, activation="relu"),
  layers.Dense(99, activation="relu"),
  layers.Dense(99, activation="relu"),
  layers.Dense(99, activation="relu"),
  layers.Dense(7, activation="softmax")
])

model_white.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),
            optimizer=tf.keras.optimizers.Adam(),
            metrics=['accuracy'])
model_white.fit(winequality_white_features, winequality_white_labels,
              epochs=200, verbose=0,
              callbacks=[tensorboard_callback_white])
print("Elapsed: " + str(time.perf_counter() - t))


# Binary classification
print("Binary classification")
# Note that 3 was already subtracted above, so 5=excellent, 4=good
winequality_red_labels_bin = winequality_red_labels.copy()
for i in range(len(winequality_red_labels_bin)):
  if winequality_red_labels_bin[i] >= 4:
    winequality_red_labels_bin[i] = 1
  else:
    winequality_red_labels_bin[i] = 0

winequality_white_labels_bin = winequality_white_labels.copy()
for i in range(len(winequality_white_labels_bin)):
  if winequality_white_labels_bin[i] >= 4:
    winequality_white_labels_bin[i] = 1
  else:
    winequality_white_labels_bin[i] = 0
```

```
# Number of trials to perform
trials = 4
for trial in range(trials):
  print("== Start trial " + str(trial))

  # Red
  print("Building red model")
  t = time.perf_counter()
  model_red = tf.keras.Sequential([
    layers.Dense(11, activation="relu"),
    layers.Dense(99, activation="relu"),
    layers.Dense(99, activation="relu"),
    layers.Dense(1, activation="sigmoid")
  ])
  model_red.compile(loss=tf.keras.losses.BinaryCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(),
              metrics=['accuracy'])
  model_red.fit(winequality_red_features, winequality_red_labels_bin,
                epochs=200, verbose=0,
                callbacks=[tensorboard_callback_red])
  print("Elapsed: " + str(time.perf_counter() - t))

  # White
  print("Building white model")
  t = time.perf_counter()
  model_white = tf.keras.Sequential([
    layers.Dense(11, activation="relu"),
    layers.Dense(99, activation="relu"),
    layers.Dense(99, activation="relu"),
    layers.Dense(1, activation="sigmoid")
  ])

  model_white.compile(loss=tf.keras.losses.BinaryCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(),
              metrics=['accuracy'])
  model_white.fit(winequality_white_features, winequality_white_labels_bin,
                epochs=200, verbose=0,
                callbacks=[tensorboard_callback_white])
  print("Elapsed: " + str(time.perf_counter() - t))
```

```
# Simplified model
# Lowest correlation for both datasets:
#  - residual sugar, free sulfur dioxide, pH, fixed acidity, total sulfur dioxide
'''
cut_list = ["residual sugar", "free sulfur dioxide", "pH", "fixed acidity", "total sulfur dioxide"]
for cut in cut_list:
  winequality_red_features.drop(cut, axis=1)
  winequality_white_features.drop(cut, axis=1)

# Classification simple
print("Classification simple")
# Number of trials to perform
trials = 0
for trial in range(trials):
  print("== Start trial " + str(trial))

  # Red
  print("Building red model")
  t = time.perf_counter()
  model_red = tf.keras.Sequential([
    layers.Dense(6, activation="relu"),
    layers.Dense(99, activation="relu"),
    layers.Dense(99, activation="relu"),
    layers.Dense(99, activation="relu"),
    layers.Dense(99, activation="relu"),
    layers.Dense(99, activation="relu"),
    layers.Dense(7, activation="softmax")
  ])
  model_red.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(),
              metrics=['accuracy'])
  model_red.fit(winequality_red_features, winequality_red_labels,
                epochs=200, verbose=0,
                callbacks=[tensorboard_callback_red])
  print("Elapsed: " + str(time.perf_counter() - t))

  # White
  print("Building white model")
  t = time.perf_counter()
  model_white = tf.keras.Sequential([
    layers.Dense(6, activation="relu"),
    layers.Dense(99, activation="relu"),
    layers.Dense(99, activation="relu"),
    layers.Dense(99, activation="relu"),
    layers.Dense(99, activation="relu"),
    layers.Dense(99, activation="relu"),
    layers.Dense(7, activation="softmax")
  ])

  model_white.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              optimizer=tf.keras.optimizers.Adam(),
              metrics=['accuracy'])
  model_white.fit(winequality_white_features, winequality_white_labels,
                epochs=200, verbose=0,
                callbacks=[tensorboard_callback_white])
  print("Elapsed: " + str(time.perf_counter() - t))

'''
```

```
# Evaluate
print()
print("Evaluate")
print("White model, white data")
model_white.evaluate(winequality_white_features, winequality_white_labels)
print("White model, red data")
model_white.evaluate(winequality_red_features, winequality_red_labels)
print("Red model, white data")
model_red.evaluate(winequality_white_features, winequality_white_labels)
print("Red model, red data")
model_red.evaluate(winequality_red_features, winequality_red_labels)

print()
print("Evaluate binary")
print("White model, white data")
model_white.evaluate(winequality_white_features, winequality_white_labels_bin)
print("White model, red data")
model_white.evaluate(winequality_red_features, winequality_red_labels_bin)
print("Red model, white data")
model_red.evaluate(winequality_white_features, winequality_white_labels_bin)
print("Red model, red data")
model_red.evaluate(winequality_red_features, winequality_red_labels_bin)

# Display
print("Display results")
%tensorboard --logdir logs/fit
```