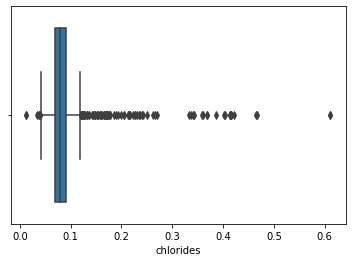
Section II – EDA

We made the assumption that the data collection process was sound and did not question the manner in which it was accumulated and recorded. That part of the process is outside the scope of both this assignment and class. Furthermore, the dataset has been widely used by many others which further supports our assumption. The records were already in a single CSV file format, so we didn’t need to combine any files. This made it very easy to import using the pandas API in Python. By bringing in the file as a pandas dataframe, we were able to maipulate and explore very quickly. We found that there were 1,599 entries each representing a different wine variety. For each one, there were 12 columns of information, 11 which were features and one final “target” column indicating quality.

Each of the feature columns was a float datatype, which is important for the algorithms later. Since none of the features were categorical, or even binary, we were enthusiastic that some good patterns would emrege. The target column of quality was in an integer format, theoretically from 0-10. However, we observed only records from 3-8. This made the data relatively concentrated near the center, which creates a challenge in terms of classing wines. In fact, the distribution centers heavily around those wines at either a 5 or a 6. So even if a classification algorithm were to be very close and guess a 6, when in reality the wine was a 5, that would adversely impact accuracy.

The data itself is very complete and workable. None of the columns had missing values or excessive zeros. Generally, excessive zeros indicate that a record is missing and was just given a ‘0’. We tested this formulaically with a simple python script. By simply summing up the total records for each feature with a zero, we could quickly see the results. We found sort of an odd outcome: one column had about 10% of the records with a zero for that field. This is unusual for the dataset at large since not one other column had a zero. We did not see any missing values either, so for there to be a 10% error in only field did not seem right. In order to help gain comfort, we did some research and found that there are indeed wines that contain zero citric acid. Individuals who are sensitive to acidity may select one of these alternatives to help alleviate the problem. This is more of a unique situation, the exception rather than the norm. Thus if 10% of our data had zero citric acid, this is most likely a valid result. Therefore, we have decided to keep all the data without making alterations for these zeros.

Continuing on with our data check, we investigated the possibility of outliers. Here we wanted to again see if any data had been erroneously entered or recorded and thus skew the results. Given we have a smaller set of records to work with, the presence of outliers could have a greater impact on the models. As we had the luxury of each feature column being numeric, we ran a box-and-whisker plot for each one and visually inspected each one. While most fit the classic structure, we did see some that, at first glance looked like there may be outliers. However, when we considered the scale of each one, we did not conclude that there were any misrecorded data points. For example consider the results of one of the attributes, chloride, shown below in Figure 1. While it may first appear that there are multiple outliers, the scale shows how close these entries are. We see this as a valid set and do not intend to remove any potentially outliers. We believe that keeping the records is more beneficial.

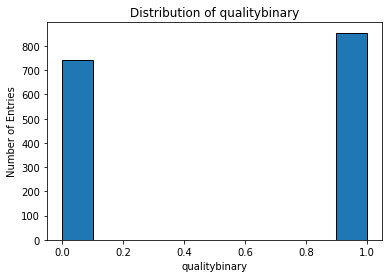
 *Figure 1: chlorides*

We also checked for duplicate records, again using Python, and unsurprisingly found none. After going through these different tests, we have gained a great deal of comfort with the dataset and its accuracy.

Next, to help guide our approaches, we ran a general correlation matrix on the entire set. Here we employed some of the built-in functions of pandas, such as the correlation, and plotted the results. Similar to inspecting the outliers visually, it was helpful to see where there may be some correlations to investigate. Had our data and defined problem been a bit more ambiguous, we would have dug in further here. Fortunately, we already had a clear target variable so we wanted to start working with the most meaningful relationships. We used pandas again to check the correlation of each feature with the target (quality). We saw some relatively strong correlations, with alcohol at almost .5 (out of 1). There were also some negative correlations as well that we want to include as well. Our big takeaway was that we had some correlations to work with, but none that were too strong. With highly correlated data, the risk of having a poorly trained model becomes greater. After checking, we do feel that any columns should be removed due to an overly high correlation.

Section III – Binary Classification

Our first goal was to simply classify a wine as either “good” or “bad”. To create a binary target, we split the wines roughly in half: those with a recorded quality of less than 6 were considered bad, and those with 6 or greater were considered good. Our new field we labeled as “qualitybinary”, and the distribution, shown below, gave us a nice and balanced target.

 *Figure 2: Quality distribution*

On the other hand, as mentioned above, most records were either a 5 or 6. So the differences between the two are very slim. As such, this will make for a greater challenge for the algorithms.

The most important thing we tried to control for was consistency across approaches. By standardizing the train/test split, and the way the functions interact with the data, we can be sure to capture consistent results. We set up a “driver” function where we can pass a model and get back the relevant metrics. This also allows for easier document flow.

Our binary classification approaches were Naive Bayes, Decesion Tree, Gradient Boosting, and Random Forest. We have discussed all of these throughout the class, and all are appropriate for this type of data. We wanted to test a couple of different things here to try and find the optimal model. Looking at hyperparameters is of course one aspect, but we did some specific tuning that fit with each algorithm.

Beginning with the simplest approach, Naive Bayes, we gave ourselves a baseline. Because Naive Bayes is the least sophisticated algorithm, we assumed it to give the least accurate results. If we use the most basic classification algorithm with no tuning, then each of the next models should do better. If not, then something is certainly wrong and would cause for additional investigation. We also chose to assume Gaussian distribution, as that is statistically the most likely outcome. Additionally, every single feature we tested was numeric. So Guassian Naive Bayes is the most appropriate choice. Without any tuning, we achived ~70.4% accuracy, which is decent, but also ended up being the worst peforming algorithm. In this way, it achieved exactly what we wanted by providing a simple baseline.

Next we ran a decision tree classifier. This is also a relatively unsophisticated algorithm, but allows for lots of hyperparameter tuning. In order to optimize this approach, we ran a cross-validation grid search. By creating a pipeline of variables to test, we could allow the cross-validation score to determine which ones to ultimately use. We focused on criterion (either gini or entropy) and the max depth. The result of some heavy computations indicated that gini and a max depth of 6 yielded the best results. Indeed, a decision tree with these hyperparameters beat the Naive Bayes with a ~71.4% accuracy. We would like to note that the computational expense of achieving this optimized approach is almost not worth the small improvement in accuracy.

Moving forward with more sophistication, we tested a variant of gradient boosting next. The gradient boosting algorithm also allows for much tuning, and is also a good candidate for a grid search. Given the low improvement on the decision tree grid search, we decided to take a different approach. There is a “pre-tuned” version of gradient boosting available known as XGBoost (which stands for Extreme Gradient Boosting). Here we have a canned algorithm that should perform better than a stock gradient boosting approach, but perhaps not quite as well as a full grid search. The tradeoff is that it is very quick to implement, and gave excellent results. Without touching a single hyperparameter, the XGBoost was able to achieve ~77.1% accuracy. This far exceeds both of the other approaches, while also saving on computational time.

Finally, we tried one last approach, the random forrest. Another sophisticated algorithm, the random forrest is actually a collection of decision trees, so we wanted to see how it perform against our previously implemented decision tree. To tune, we focused on (arguably) the most important factor, n\_estimators (or number of trees). We did a mini grid-search and iterated through 20 different estimators and kept track of the best one. For our dataset, this turned out to be 14, which gave us the best accuracy of any binary classifier we tried: ~78.5%. While more computationally expensive than the XGBoost, it was still much faster than running the full grid search for the decision tree. Our loop took less than a second to run, while the grid search took over 30 seconds of near full CPU running time.

Our takeaway from this portion of the exercise was that the more sophisticated algorithms generally perform better. Also it is most likely worth tuning some hyperparameters in an effort to maximize accuracy, provided the resources are available. Regarding a benchmark, the XGBoost approach is just so quick and easy to implement that we believe that would be a better benchmark going forward. It takes no more time than a simple Naive Bayes algorithm, yet yields substantially better results. If a user can find a combination of hyperparameters for a different algorithm that beats it, then that is a good indicator of success.

Being able to classify wines as either “good” or “bad” with accuracy in the high 70’s is a decent result. However, keeping in mind that the boundary line is actually the difference between a 5 or 6 out of 10, we believe these results are slightly more impressive. Overally, we are satsfied with our binary classification approach.