FinalProject_group78

June 12, 2019

1 COGS 108 Final Project

2 Overview

For our project, we decided to see if we could predict the quality of wine given its ingredients. Since wine is so easily and heavily influenced by a number of factors that we can't control for (i.e humidity, grape quality, etc), we wanted to focus on its internal workings. We hypothesized that the volume of certain features, namely alcohol percentage, volatile acidity and citric acid, were crucial in determining a wine's quality rating. Using two datasets to test our prediction, we found that these, as well as the other characteristics, had no individual effect on overall rating, but rather, the rating was derived from a combination of different features.

3 Group Members - Names and ID

- Alysha Ali (A15345048)
- Tsai-Ying Ying Chuang (A15215376)
- Zheng Zeng (A14679117)
- Melody Yu (A14599481)
- Ai-Ting Hsieh (A13595395)

4 Part 1 - Introduction & Background

Since its creation 6000 years ago, wine has been adapted widely — from an instrument for religious ceremonies to a bargaining tool between countries (https://vinepair.com/wine-colonized-world-wine-history/#5). Despite the variety in usage, wine has constantly been regarded as a social device and has only risen in popularity (https://www.svb.com/globalassets/library/images/svb-2018-wine-report.pdf). That's why, with the combination of the quickly-approaching legal drinking age for several of our group members and its increasing appearance in social media(https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4432862/), we thought it was important to determine how the quality of a wine is determined. Specifically, we aim to discover exactly what ingredients are crucial in deriving a wine's quality rating.

We came to our hypothesis by first researching wine characteristics (https://winefolly.com/review/wine-characteristics/). From there, we saw that certain elements, like citric acid content and volatile acidity, directly influenced a wine's sweetness. From there, we looked at ranges and eliminated factors like water density, which only ranged from 0.99

to 1.00 and therefore could not have as large an impact on quality as other factors, like total sulfur dioxide for example, which had a range of over 200.

This information is important for all consumers, from pre-teens getting their first sip of wine to experienced connoisseurs and businesses. With this data analysis, instead of spending hours trying to choose a wine everyone will love, party hosts and other entertainers will know what the most favored wine features are and be able to buy a delicious spirit in a snap. For instance, knowing what chemicals make a wine popular will reduce how much time winemakers spend in perfecting their formula and ultimately maximize efficiency and increase profit in the long run. Thus, this information can affect more than just consumers.

Since these datasets were previously the subject of a competition hosted by the University of California, Irvine, there have been several other projects that have analyzed similar questions and/or have used the same datasets. Most notably, is another study that also attempted to predict the quality of wine (https://www.kaggle.com/vishalyo990/prediction-of-quality-of-wine). However, they differed in their methods used and the overall accuracy of their machine learning model.

4.1 Our question

Can we predict the quality of a wine given a list of its physicochemical features?

4.2 What we think

We hypothesize that we can predict the quality of the wine by its individual features. Especially, we predict that the lower volatile acidity, medium alcohol percentage, and higher citric acid will create the best quality of the wine.

5 Part 2 - Data Wrangling

To answer our question, we will be using two datasets: one on red wines and one on white wines (source: https://archive.ics.uci.edu/ml/datasets/wine+quality). The red wine dataset had 1601 observations, while the white wine dataset had 4900. These datasets are identical in the features they analyze, namely a list of physiochemical ingredients in wine:

- fixed acidity - most acids involved with wine or fixed or nonvolatile (do not evaporate readily)
- volatile acidity - the amount of acetic acid in wine, which at too high of levels can lead to an unpleasant, vinegar taste
- citric acid - found in small quantities, citric acid can add 'freshness' and flavor to wines
- residual sugar - the amount of sugar remaining after fermentation stops, it's rare to find wines with less than 1 gram/liter and wines with greater than 45 grams/liter are considered sweet
- chlorides - the amount of salt in the wine
- free sulfur dioxide - the free form of SO2 exists in equilibrium between molecular SO2 (as a dissolved gas) and bisulfite ion; it prevents microbial growth and the oxidation of wine

- total sulfur dioxide - amount of free and bound forms of S02; in low concentrations, SO2 is mostly undetectable in wine, but at free SO2 concentrations over 50 ppm, SO2 becomes evident in the nose and taste of wine
- density - the density of water is close to that of water depending on the percent alcohol and sugar content
- pH - describes how acidic or basic a wine is on a scale from 0 (very acidic) to 14 (very basic); most wines are between 3-4 on the pH scale
- sulphates - a wine additive which can contribute to sulfur dioxide gas (S02) levels, wich acts as an antimicrobial and antioxidant
- alcohol - the percent alcohol content of the wine
- quality - output variable (based on sensory data, score between 0 and 10)

We combined the two in several ways during our process. For instance, in one part, we began by graphing the individual features together in order to see a comparison between white and red wines. In another part, we used random samples from both datasets in order to train and test our algorithm.

5.1 Setup

```
In [4]: import pandas as pd
        import numpy as np
        import pylab
        import requests
        import bs4
        from bs4 import BeautifulSoup
        import matplotlib.pyplot as plt
        from sklearn.neighbors import KDTree
        from sklearn.utils import shuffle
        from pandas.plotting import scatter_matrix
```

```
In [5]: #Here we import the two datasets that we will be usiing in our final project
        df_red = pd.read_csv('wineQualityReds.csv')
        df_white = pd.read_csv('wineQualityWhites.csv')

        # print out the column names
        print(df_red.columns.values)
```

```
['Unnamed: 0' 'fixed.acidity' 'volatile.acidity' 'citric.acid'
 'residual.sugar' 'chlorides' 'free.sulfur.dioxide' 'total.sulfur.dioxide'
 'density' 'pH' 'sulphates' 'alcohol' 'quality']
```

6 Part 3 - Data Cleaning, Processing & Visualization

Here, we will clean up our data before we start analyzing. We will remove rows with missing data, and delete columns that are not useful. More explanation will be included as we clean up the data.

```
In [6]: df_red[:5]
```

```
Out[6]:    Unnamed: 0  fixed.acidity  volatile.acidity  citric.acid  residual.sugar  \
        0           1            7.4              0.70         0.00             1.9
        1           2            7.8              0.88         0.00             2.6
        2           3            7.8              0.76         0.04             2.3
        3           4           11.2              0.28         0.56             1.9
        4           5            7.4              0.70         0.00             1.9

           chlorides  free.sulfur.dioxide  total.sulfur.dioxide  density    pH  \
        0      0.076                 11.0                  34.0   0.9978  3.51
        1      0.098                 25.0                  67.0   0.9968  3.20
        2      0.092                 15.0                  54.0   0.9970  3.26
        3      0.075                 17.0                  60.0   0.9980  3.16
        4      0.076                 11.0                  34.0   0.9978  3.51

           sulphates  alcohol  quality
        0       0.56      9.4        5
        1       0.68      9.8        5
        2       0.65      9.8        5
        3       0.58      9.8        6
        4       0.56      9.4        5
```

```
In [7]: df_red.shape
```

```
Out[7]: (1599, 13)
```

Drop any row with missing values

```
In [8]: df_red = df_red.dropna()
```

Drop the column labeled “Unnamed: 0”

```
In [9]: df_red = df_red.drop(['Unnamed: 0'], axis=1)
```

```
In [10]: df_red.shape
```

```
Out[10]: (1599, 12)
```

```
In [11]: df_red[:5]
```

```
Out[11]:    fixed.acidity  volatile.acidity  citric.acid  residual.sugar  chlorides  \
         0            7.4              0.70         0.00             1.9      0.076
         1            7.8              0.88         0.00             2.6      0.098
```

```
2              7.8              0.76         0.04             2.3      0.092
3             11.2              0.28         0.56             1.9      0.075
4              7.4              0.70         0.00             1.9      0.076

   free.sulfur.dioxide  total.sulfur.dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol  quality
0      9.4        5
1      9.8        5
2      9.8        5
3      9.8        6
4      9.4        5
```

```
In [12]: df_white[:5]
```

```
Out[12]:    Unnamed: 0  fixed.acidity  volatile.acidity  citric.acid  residual.sugar  \
         0           1            7.0              0.27         0.36            20.7
         1           2            6.3              0.30         0.34             1.6
         2           3            8.1              0.28         0.40             6.9
         3           4            7.2              0.23         0.32             8.5
         4           5            7.2              0.23         0.32             8.5

            chlorides  free.sulfur.dioxide  total.sulfur.dioxide  density    pH  \
         0      0.045                 45.0                 170.0   1.0010  3.00
         1      0.049                 14.0                 132.0   0.9940  3.30
         2      0.050                 30.0                  97.0   0.9951  3.26
         3      0.058                 47.0                 186.0   0.9956  3.19
         4      0.058                 47.0                 186.0   0.9956  3.19

            sulphates  alcohol  quality
         0       0.45      8.8        6
         1       0.49      9.5        6
         2       0.44     10.1        6
         3       0.40      9.9        6
         4       0.40      9.9        6
```

```
In [13]: df_white.shape
```

```
Out[13]: (4898, 13)
```

Drop any rows with missing values

```
In [14]: df_white = df_white.dropna()
```

Drop the column labeled as “Unnamed: 0”

```
In [15]: df_white = df_white.drop(['Unnamed: 0'], axis=1)
```

```
In [16]: df_white.shape
```

```
Out[16]: (4898, 12)
```

```
In [17]: df_white[:5]
```

```
Out[17]:    fixed.acidity  volatile.acidity  citric.acid  residual.sugar  chlorides  \
         0            7.0              0.27         0.36            20.7      0.045
         1            6.3              0.30         0.34             1.6      0.049
         2            8.1              0.28         0.40             6.9      0.050
         3            7.2              0.23         0.32             8.5      0.058
         4            7.2              0.23         0.32             8.5      0.058

            free.sulfur.dioxide  total.sulfur.dioxide  density    pH  sulphates  \
         0                 45.0                 170.0   1.0010  3.00       0.45
         1                 14.0                 132.0   0.9940  3.30       0.49
         2                 30.0                  97.0   0.9951  3.26       0.44
         3                 47.0                 186.0   0.9956  3.19       0.40
         4                 47.0                 186.0   0.9956  3.19       0.40

            alcohol  quality
         0      8.8        6
         1      9.5        6
         2     10.1        6
         3      9.9        6
         4      9.9        6
```

We want to predict wine quality based on their ingredients therefore the information we need for each wine instance is the list of ingredient values and the quality. In the above cells, we first dropped any rows with missing values to ensure that we are using a dataset that contains exactly what we need. Since this is already a very clean dataset, none of the rows were dropped. The column labeled "Unnamed: 0" does not seem to be useful at all since we also have the index of each row. Therefore, we dropped the entire column. Now, the resulting dataframe only contains information we need.

Since the white wine dataset contains more instances, we will only take 1599 instances from it randomly to match the number of instances of red wine.

```
In [18]: df_white = shuffle(df_white).reset_index(drop = True)

         df_white = df_white[:1599]
```

Here we will combine two wine sets.

```
In [19]: # appending datasets of both white wine and red wine
         df_combined = df_red.append(df_white, ignore_index =True)
         df_combined = shuffle(df_combined)
         df_combined = df_combined.reset_index(drop=True)

         df_combined.shape
```

```
Out[19]: (3198, 12)
```

```
In [20]: desc = df_combined.describe()
         desc
```

```
Out[20]:       fixed.acidity  volatile.acidity  citric.acid  residual.sugar  \
         count    3198.000000       3198.000000  3198.000000     3198.000000
         mean        7.593340          0.403094     0.302489        4.489290
         std         1.549998          0.192557     0.164095        4.192491
         min         4.200000          0.080000     0.000000        0.600000
         25%         6.600000          0.250000     0.210000        1.900000
         50%         7.200000          0.360000     0.300000        2.400000
         75%         8.200000          0.530000     0.400000        6.000000
         max        15.900000          1.580000     1.000000       31.600000

                  chlorides  free.sulfur.dioxide  total.sulfur.dioxide      density  \
         count  3198.000000          3198.000000           3198.000000  3198.000000
         mean      0.066540            25.200281             91.969669     0.995411
         std       0.042143            16.418179             59.627286     0.002819
         min       0.012000             1.000000              6.000000     0.987420
         25%       0.043000            12.000000             37.000000     0.993443
         50%       0.059000            22.000000             88.000000     0.995800
         75%       0.080000            35.000000            136.000000     0.997400
         max       0.611000           112.000000            366.500000     1.010300

                         pH    sulphates      alcohol      quality
         count  3198.000000  3198.000000  3198.000000  3198.000000
         mean      3.249912     0.572745    10.461024     5.761101
         std       0.163454     0.167567     1.151039     0.851406
         min       2.720000     0.230000     8.000000     3.000000
         25%       3.140000     0.460000     9.500000     5.000000
         50%       3.240000     0.550000    10.200000     6.000000
         75%       3.360000     0.650000    11.200000     6.000000
         max       4.010000     2.000000    14.900000     8.000000
```

```
In [21]: corrs = df_combined.corr()
         corrs
```

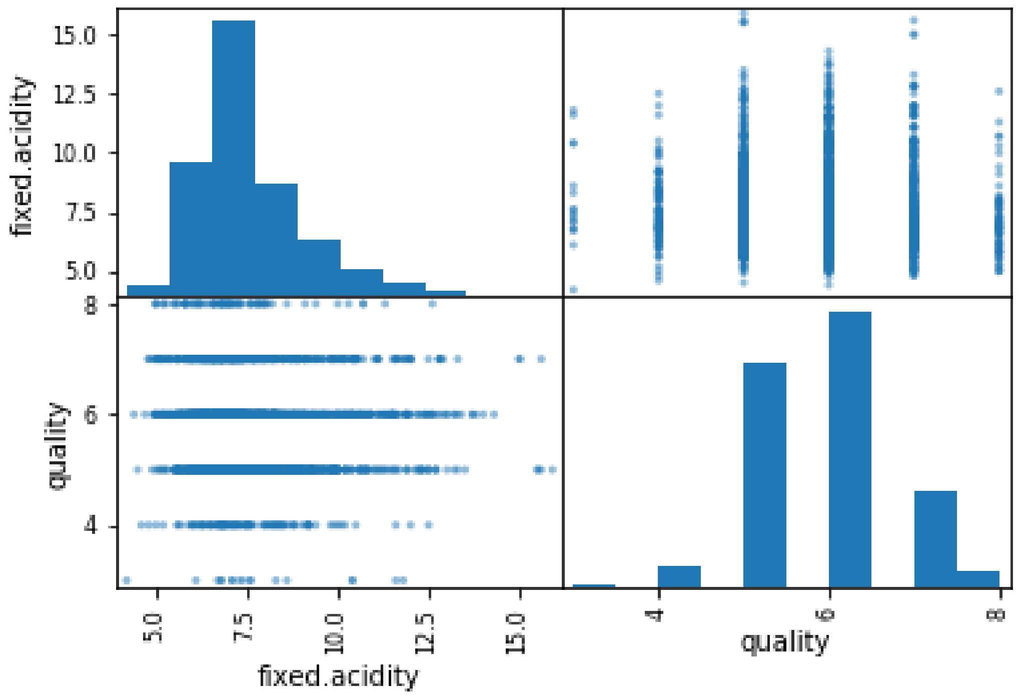
```
Out[21]:                       fixed.acidity  volatile.acidity  citric.acid  \
         fixed.acidity              1.000000          0.171727     0.407795
         volatile.acidity           0.171727          1.000000    -0.458312
         citric.acid                0.407795         -0.458312     1.000000
         residual.sugar            -0.164296         -0.281011     0.164973
         chlorides                  0.296037          0.361918     0.059043
         free.sulfur.dioxide       -0.334627         -0.397302     0.132845
         total.sulfur.dioxide      -0.372436         -0.454174     0.197656
         density                    0.556108          0.324357     0.111073
         pH                        -0.292510          0.331062    -0.424707
         sulphates                  0.345303          0.200511     0.113168
```

```
alcohol                   -0.090773         -0.095184     0.034802
quality                   -0.043420         -0.333890     0.158044

                     residual.sugar  chlorides  free.sulfur.dioxide  \
fixed.acidity             -0.164296   0.296037            -0.334627
volatile.acidity          -0.281011   0.361918            -0.397302
citric.acid                0.164973   0.059043             0.132845
residual.sugar             1.000000  -0.195536             0.489241
chlorides                 -0.195536   1.000000            -0.257042
free.sulfur.dioxide        0.489241  -0.257042             1.000000
total.sulfur.dioxide       0.555424  -0.326883             0.768073
density                    0.349580   0.376899            -0.092566
pH                        -0.293666   0.018465            -0.180862
sulphates                 -0.236341   0.462941            -0.243341
alcohol                   -0.280145  -0.227448            -0.152875
quality                    0.023986  -0.193329             0.090369

                     total.sulfur.dioxide   density        pH  sulphates  \
fixed.acidity                   -0.372436  0.556108 -0.292510   0.345303
volatile.acidity                -0.454174  0.324357  0.331062   0.200511
citric.acid                      0.197656  0.111073 -0.424707   0.113168
residual.sugar                   0.555424  0.349580 -0.293666  -0.236341
chlorides                       -0.326883  0.376899  0.018465   0.462941
free.sulfur.dioxide              0.768073 -0.092566 -0.180862  -0.243341
total.sulfur.dioxide             1.000000 -0.136670 -0.296915  -0.333072
density                         -0.136670  1.000000  0.022438   0.326821
pH                              -0.296915  0.022438  1.000000   0.139735
sulphates                       -0.333072  0.326821  0.139735   1.000000
alcohol                         -0.202789 -0.618483  0.144421   0.017505
quality                          0.007226 -0.282348 -0.014176   0.066996

                      alcohol   quality
fixed.acidity        -0.090773 -0.043420
volatile.acidity     -0.095184 -0.333890
citric.acid           0.034802  0.158044
residual.sugar       -0.280145  0.023986
chlorides            -0.227448 -0.193329
free.sulfur.dioxide  -0.152875  0.090369
total.sulfur.dioxide -0.202789  0.007226
density              -0.618483 -0.282348
pH                    0.144421 -0.014176
sulphates             0.017505  0.066996
alcohol               1.000000  0.439750
quality               0.439750  1.000000
```

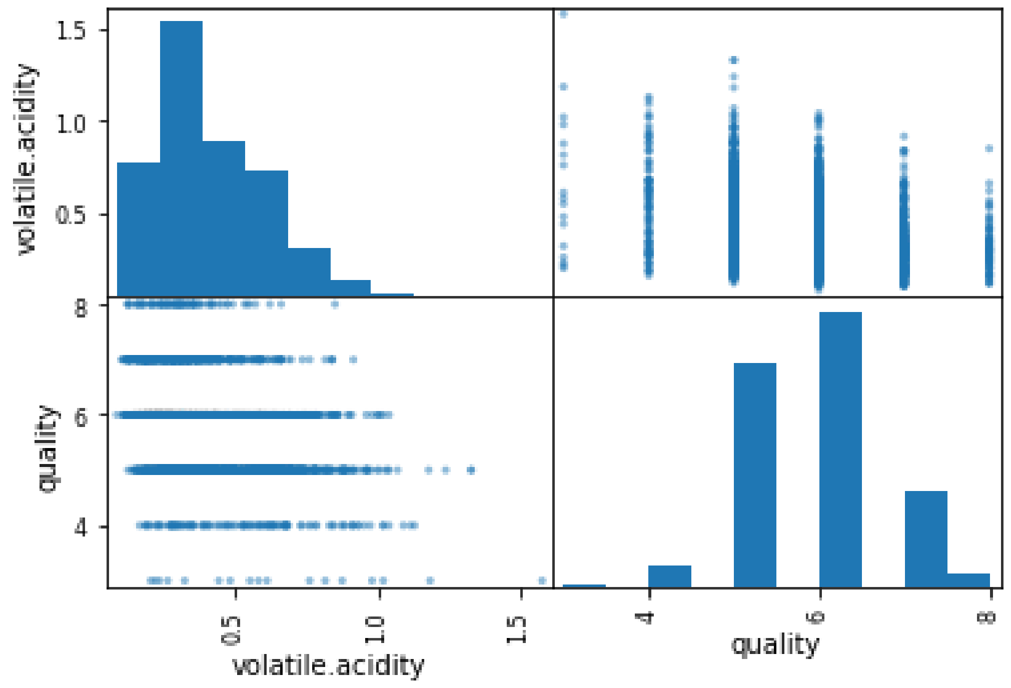
By reading the above correlation table, it seems like none of the feature correlates with the wine's quality, we will plot out some matrices to see if that really is the case.

```
In [22]: scatter_matrix(df_combined[['fixed.acidity','quality']])
```

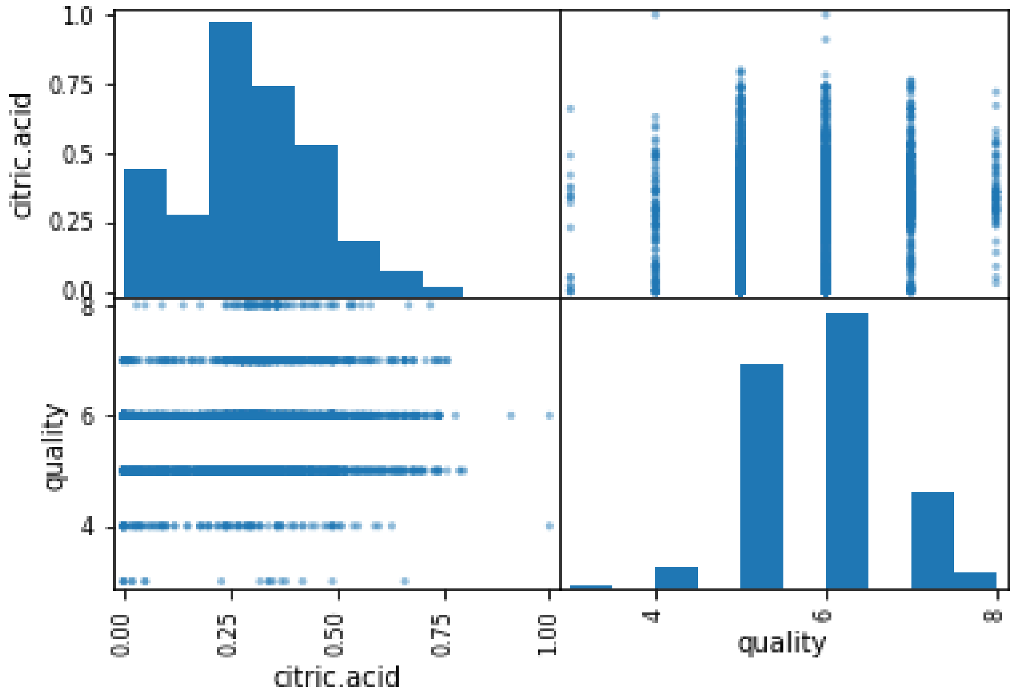
```
f4 = plt.gcf()
```



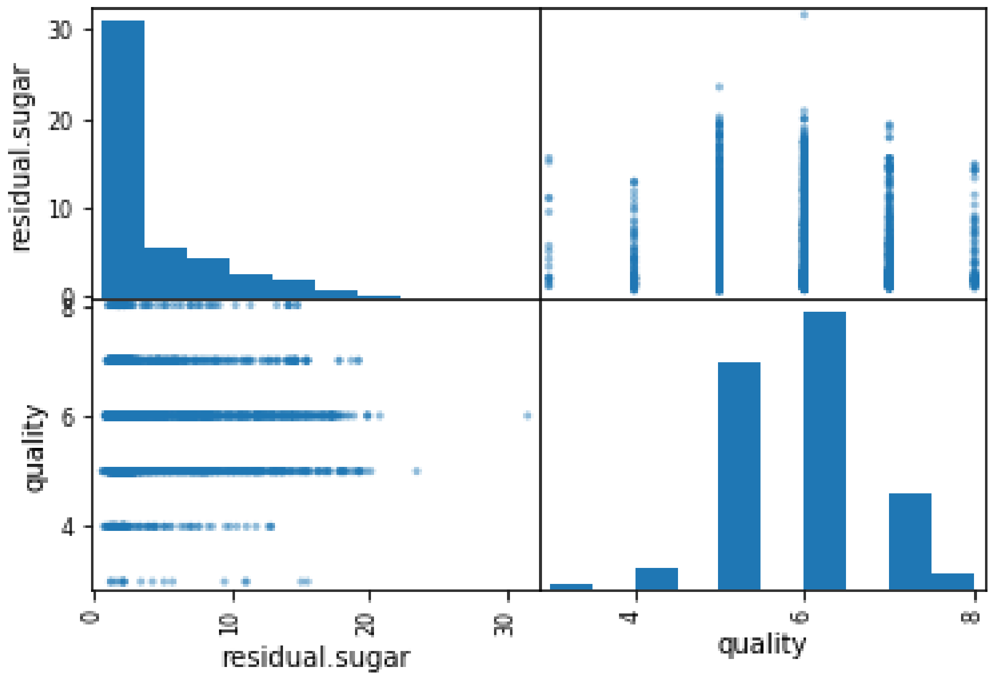
```
In [23]: scatter_matrix(df_combined[['volatile.acidity','quality']])

         f4 = plt.gcf()
```



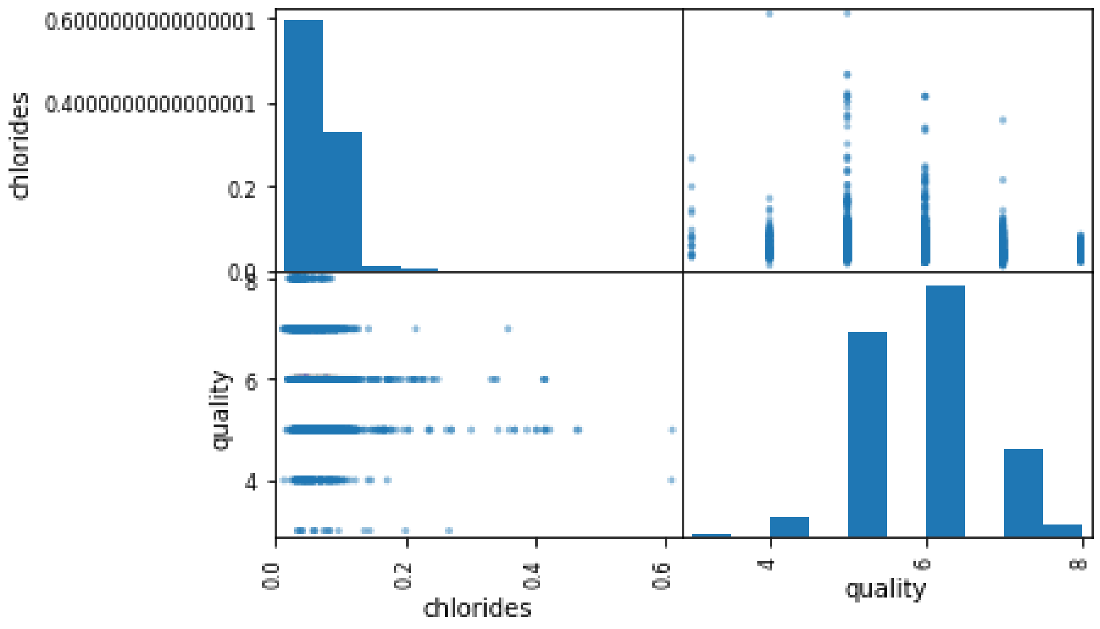
```
In [24]: scatter_matrix(df_combined[['citric.acid','quality']])

         f4 = plt.gcf()
```



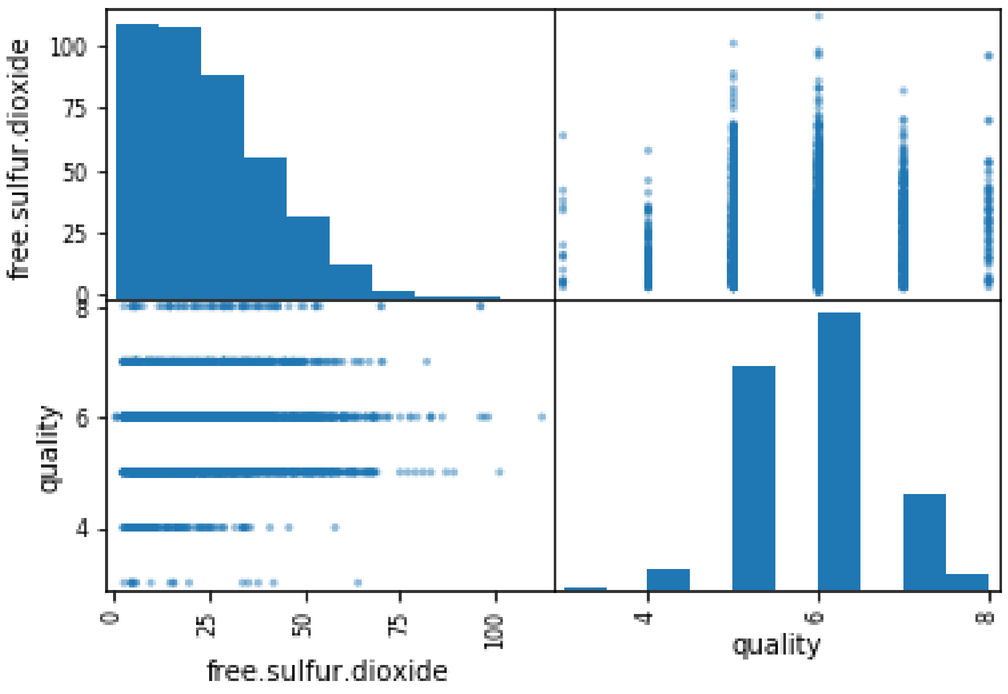
```
In [25]: scatter_matrix(df_combined[['residual.sugar','quality']])

         f4 = plt.gcf()
```



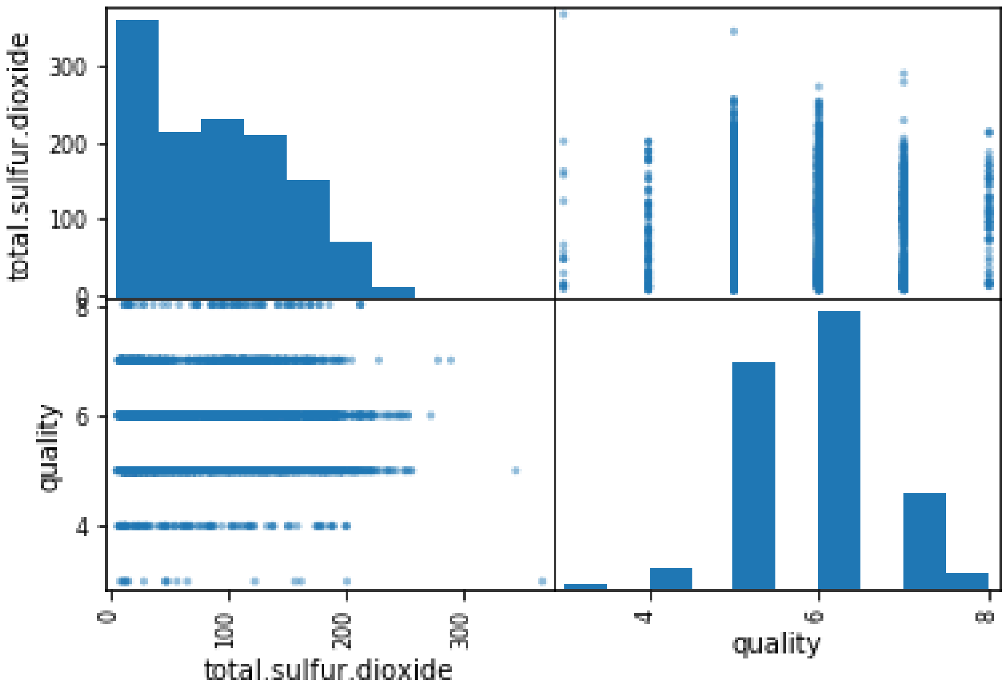
```
In [26]: scatter_matrix(df_combined[['chlorides','quality']])

         f4 = plt.gcf()
```



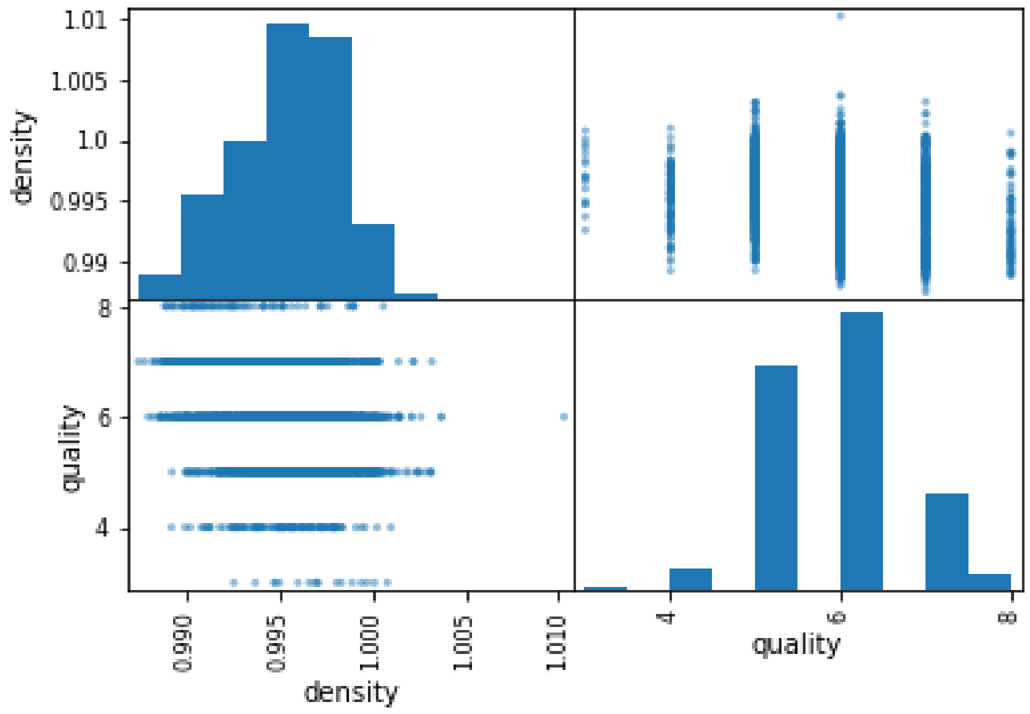
```
In [27]: scatter_matrix(df_combined[['free.sulfur.dioxide','quality']])

         f4 = plt.gcf()
```



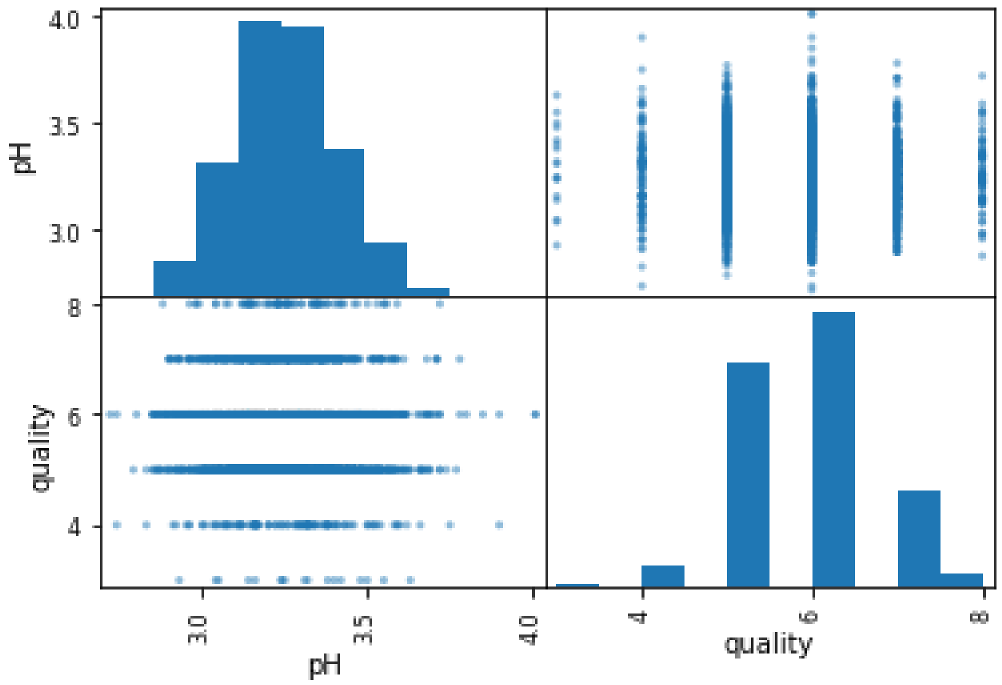
```
In [28]: scatter_matrix(df_combined[['total.sulfur.dioxide','quality']])

         f4 = plt.gcf()
```



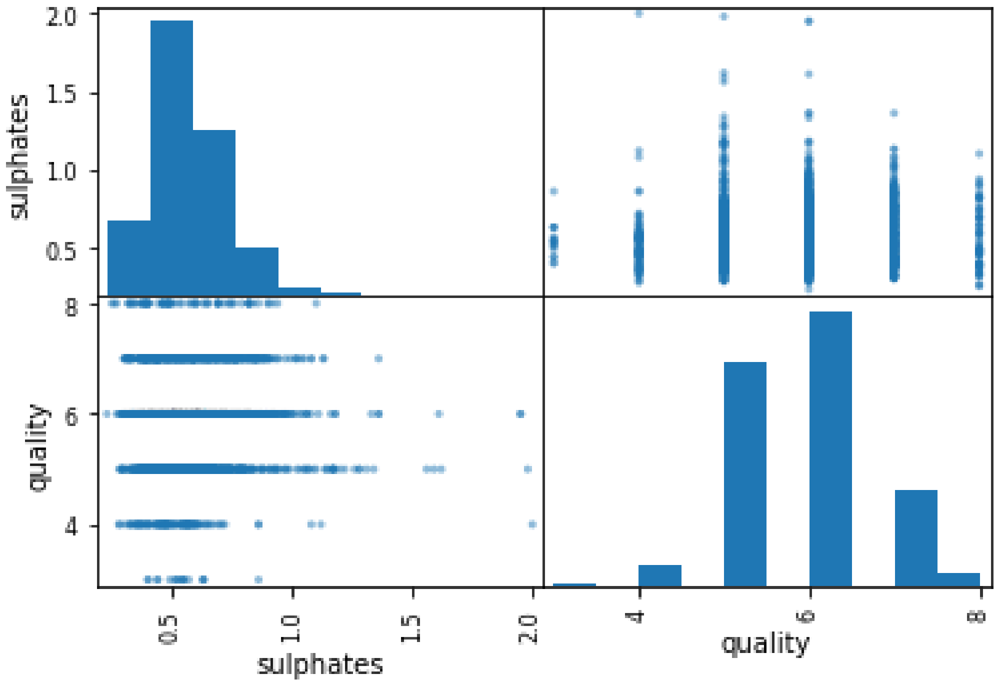
```
In [29]: scatter_matrix(df_combined[['density','quality']])

         f4 = plt.gcf()
```



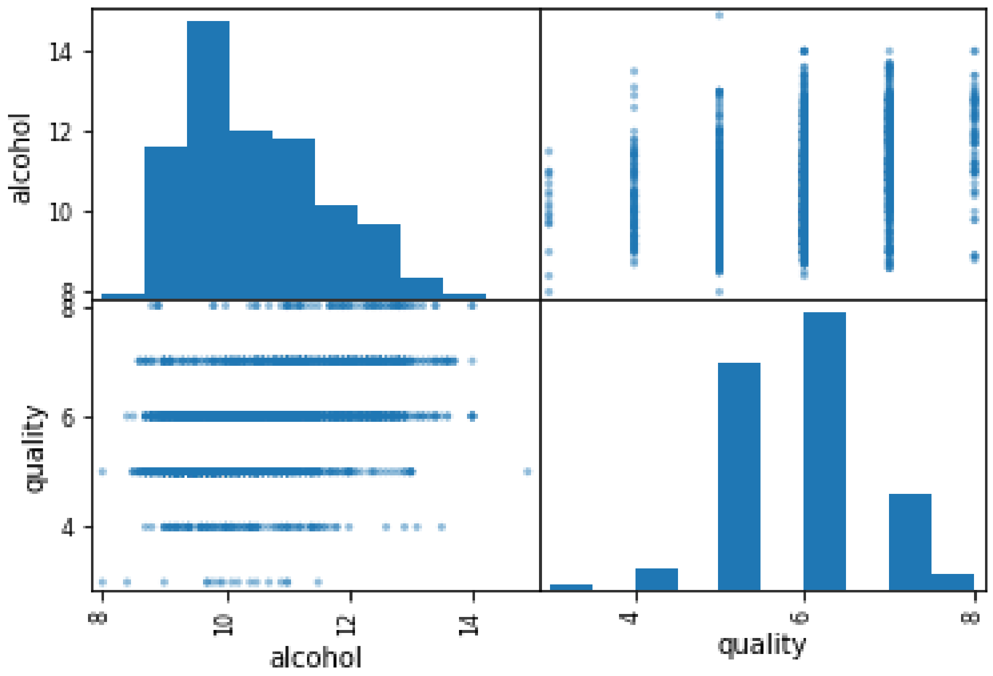
```
In [30]: scatter_matrix(df_combined[['pH','quality']])

         f4 = plt.gcf()
```



```
In [31]: scatter_matrix(df_combined[['sulphates','quality']])

         f4 = plt.gcf()
```



```
In [32]: scatter_matrix(df_combined[['alcohol','quality']])

         f4 = plt.gcf()
```



Since none of the feature seems to be highly correlated to the quality, we want to combine them all and see if doing so would allow us to predict the wine rating more accurately

```
In [33]: df_combined.shape
```

```
Out[33]: (3198, 12)
```

Similar to A5, we will split our dataset into training set (80% of total) and testing set (20% of total).

```
In [34]: num_training = int(3198*.8)
         num_training
```

```
Out[34]: 2558
```

```
In [35]: df_training = df_combined[:num_training]
         df_testing = df_combined[num_training:]
```

The following show what out training set looks like if we ignore the kind of the wine.

```
In [36]: df_training[:5]
```

```
Out[36]:    fixed.acidity  volatile.acidity  citric.acid  residual.sugar  chlorides  \
         0            6.5              0.28         0.33            15.7      0.053
         1            8.3              0.31         0.39             2.4      0.078
         2            6.8              0.18         0.37             1.6      0.055
         3            6.4              0.30         0.33             5.2      0.050
```

```
4            9.1              0.28         0.48               1.8      0.067

   free.sulfur.dioxide  total.sulfur.dioxide  density    pH  sulphates  \
0                 51.0                 190.0  0.99780  3.22       0.51
1                 17.0                  43.0  0.99444  3.31       0.77
2                 47.0                 154.0  0.99340  3.08       0.45
3                 30.0                 137.0  0.99304  3.26       0.58
4                 26.0                  46.0  0.99670  3.32       1.04

   alcohol  quality
0      9.7        6
1     12.5        7
2      9.1        5
3     11.1        5
4     10.6        6
```

Now, we will include some functions that can be useful later.

```
In [37]: #takes in a dataframe and a list of list of indices,
         #and return the predicted rate(quality) for each whine in the dataframe
         #based on the list of corresponding indices
         def predict_rate(df, inds):
             rate_predict = []
             for i in inds:
                 rateMap = {}
                 rate_mode = 0
                 for j in i:
                     curr = df.loc[j, 'quality']
                     if curr in rateMap.keys():
                         rateMap[curr]+=1
                     else:
                         rateMap[curr] = 1
                     if rateMap[curr] > rate_mode:
                         rate_mode = curr
                 rate_predict.append(rate_mode)
             return rate_predict
```

```
In [38]: #takes in a dataframe with an column containing the predicted wine quality
         #and returns the error rate
         def validation(df_after_prediction):
             error = 0
             num_row = 0
             # iterate through each row in df
             for idx, row in df_after_prediction.iterrows():
                 num_row = num_row+1
                 if not row['quality'] == row['predicted quality']:
                     error = error+1
             errorPercent = error/num_row
```

```
        #print(errorPercent)
        return errorPercent
```

```
In [39]: #takes in a dataframe
         #and return a list of rows in the dataframe without the quality column
         def get_row_list(df):
             rows_list=[]
             for idx, row in df.iterrows():
                 curr_row = [row['fixed.acidity'],row['volatile.acidity'],
                             row['citric.acid'],row['residual.sugar'],
                             row['chlorides'], row['free.sulfur.dioxide'],
                             row['total.sulfur.dioxide'], row['density'],
                             row['pH'], row['sulphates'],
                             row['alcohol']]
                 rows_list.append(curr_row)
             return rows_list
```

```
In [40]: #takes in: the training dataframe, the testing dataframe, the k value,
         #the testing dataframe's row list, a kd-tree and returns the result of validation
         def getErrorPercent(df_training, df_testing, k_value, rows_list, tree):
             inds = tree.query(rows_list, k=k_value, return_distance = False)
             df_col_pred = predict_rate(df_training, inds)
             df_testing['predicted quality'] = df_col_pred
             return validation(df_testing)
```

We want to use the KD-tree in the sklearn library to predict the quality of a wine given its list of features(ingredients). A K-Dimensional tree is a tree data structure for organizing sample points in a k dimensional space. In out project, we have 11 dimensions as there are 11 features for each wine. A KDT will store our sample points (the rows of wine) according to all their features. Therefore, by querying the closest "neighbors" of a wine from the KDT, we get a list of other wines in the KDT positioned the closest to the query point. Since we have information about those wines' quality, we can then predict the query wine's quality by taking the quality with the highest occurence among the nearest neighbors.

Now, we will build the KDT using the training dataset.

```
In [41]: #list of rows of the training dataframe
         rows_list_training = get_row_list(df_training)

         #the KD Tree that stores all the wines in the training set
         tree = KDTree(rows_list_training)
```

After building the KD tree, we want to know how many numbers of neighbors we should query from it in order to get the most accurate prediction. Out initial guess is 5, but we want to actually go through more possible values to make sure we choose a best k value.

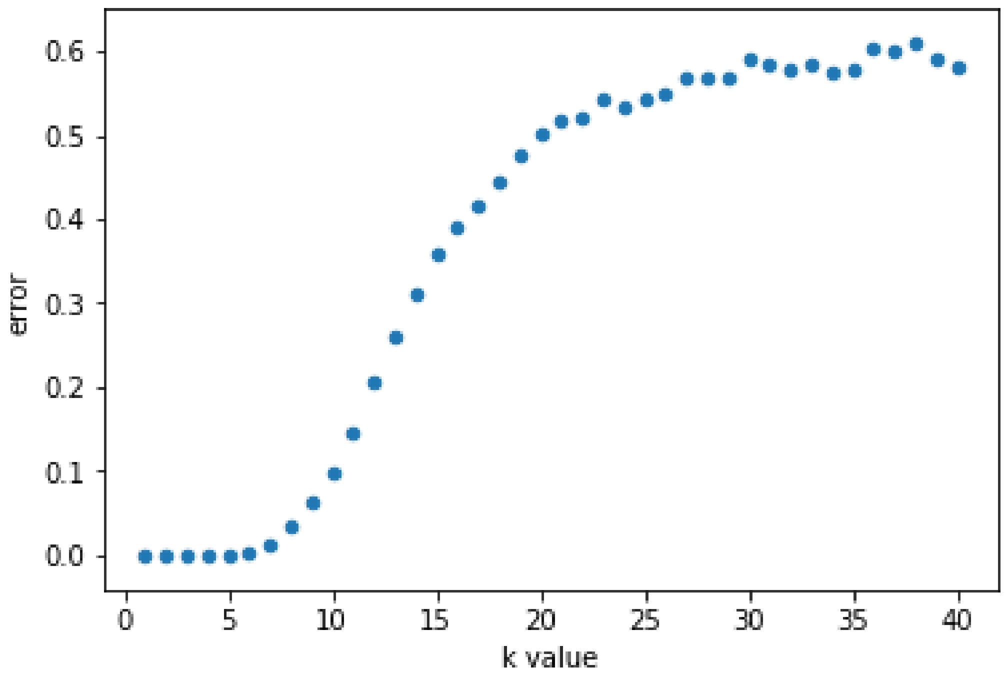
```
In [42]: #going through possible k values [1,40]
         k_val = np.arange(1, 41, 1)
         err_val=[]
         for i in k_val:
             df_training_cpy = df_training.copy()
             err_val.append(getErrorPercent(df_training, df_training_cpy,
                                            i ,rows_list_training, tree))
         df_err = pd.DataFrame()
         df_err['k value'] = k_val
         df_err['error'] = err_val
```

```
In [43]: df_err[:10]
```

```
Out[43]:    k value     error
        0         1  0.000000
        1         2  0.000000
        2         3  0.000000
        3         4  0.000000
        4         5  0.000000
        5         6  0.003518
        6         7  0.013292
        7         8  0.034011
        8         9  0.061376
        9        10  0.097733
```

```
In [44]: df_err
         df_err.plot.scatter(x='k value', y='error')
```

```
Out[44]: <matplotlib.axes._subplots.AxesSubplot at 0x1a0d745320>
```



After checking through a set of k value candidates, we decide that k=5 would be the best choice. Because it gives us a very low error. Although k values less than 5 seem to have even lower error rates, there are a few reasons why we did not choose them:

1) We do not want to choose an even k value becuase we will then need to worry about having a tie.

2) k=1 and k=3 are odd numbers, but still not nice choices as they are too small.

We will now query 5 nearest neighbors for each wine instance in our training data set, predict the quality, and add to the dataframe.

```
In [45]: #the list of rows of the training dataset
         rows_list_training = get_row_list(df_training)

         #the list of lists of nearest neighbor indices
         inds_k_5_training = tree.query(rows_list_training, k=5, return_distance = False)

         #the list of predicted quality of each wine
         df_training['predicted quality'] = predict_rate(df_training, inds_k_5_training)
```

```
/Users/aitinghsieh/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:8: SettingWithCo
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html
```

```
In [46]: df_training[:5]
```

```
Out[46]:    fixed.acidity  volatile.acidity  citric.acid  residual.sugar  chlorides  \
         0            6.5              0.28         0.33            15.7      0.053
         1            8.3              0.31         0.39             2.4      0.078
         2            6.8              0.18         0.37             1.6      0.055
         3            6.4              0.30         0.33             5.2      0.050
         4            9.1              0.28         0.48             1.8      0.067

            free.sulfur.dioxide  total.sulfur.dioxide  density    pH  sulphates  \
         0                 51.0                 190.0  0.99780  3.22       0.51
         1                 17.0                  43.0  0.99444  3.31       0.77
         2                 47.0                 154.0  0.99340  3.08       0.45
         3                 30.0                 137.0  0.99304  3.26       0.58
         4                 26.0                  46.0  0.99670  3.32       1.04

            alcohol  quality  predicted quality
         0      9.7        6                  6
         1     12.5        7                  7
         2      9.1        5                  5
         3     11.1        5                  5
         4     10.6        6                  6
```

We will compute the error percentage of our prediction.

```
In [47]: error_training = getErrorPercent(df_training, df_training, 5,
                                          rows_list_training,tree)
         error_training

/Users/aitinghsieh/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:6: SettingWithC
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.htm



Out[47]: 0.0
```

Now that our model is ready, we will go ahead and start predicting the quality of wines in the testing dataset.

```
In [48]: #the list of rows in the testing set
         rows_list_testing = get_row_list(df_testing)

         #the list of lists of nearest neighbor indices
         inds_k_5_testing = tree.query(rows_list_testing, k=5, return_distance = False)

         #the list of predicted quality of each wine
         df_testing['predicted quality'] = predict_rate(df_training, inds_k_5_testing)

/Users/aitinghsieh/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:8: SettingWithC
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.htm



In [49]: # err_testing = getErrorPercent(df_training, df_testing, 5, rows_list_testing)
         err_testing = validation(df_testing)
         err_testing

Out[49]: 0.4609375
```

7 Part 4 - Data Analysis / Results

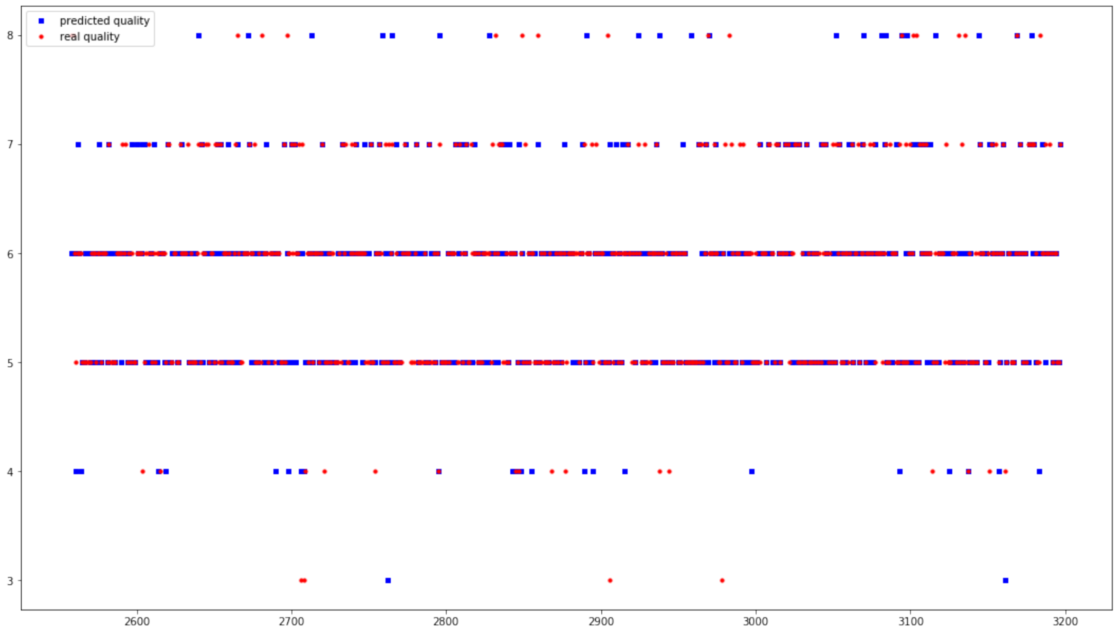
It turned out that the error percentage is pretty big, oppposite from what we were expecting. Therefore, now we want to see how in accurate our prediction is.

```
In [50]: df_testing[:10]
```

```
Out[50]:       fixed.acidity  volatile.acidity  citric.acid  residual.sugar  chloride  \
        2558            7.9              0.54         0.34            2.50      0.076
        2559            7.3              0.30         0.34            2.70      0.044
        2560            6.7              0.46         0.24            1.70      0.077
        2561            6.7              0.26         0.26            4.10      0.073
        2562            7.4              0.19         0.30            1.40      0.057
        2563            5.7              0.22         0.22           16.65      0.044
        2564           12.3              0.27         0.49            3.10      0.079
        2565            7.3              0.91         0.10            1.80      0.074
        2566            9.3              0.40         0.49            2.50      0.085
        2567            6.1              0.20         0.40            1.90      0.028

              free.sulfur.dioxide  total.sulfur.dioxide  density    pH  sulphates  \
        2558                  8.0                  17.0  0.99235  3.20       0.72
        2559                 34.0                 108.0  0.99105  3.36       0.53
        2560                 18.0                  34.0  0.99480  3.39       0.60
        2561                 36.0                 202.0  0.99560  3.30       0.67
        2562                 33.0                 135.0  0.99300  3.12       0.50
        2563                 39.0                 110.0  0.99855  3.24       0.48
        2564                 28.0                  46.0  0.99930  3.20       0.80
        2565                 20.0                  56.0  0.99672  3.35       0.56
        2566                 38.0                 142.0  0.99780  3.22       0.55
        2567                 32.0                 138.0  0.99140  3.26       0.72

              alcohol  quality  predicted quality
        2558     13.1        8                  6
        2559     12.8        8                  6
        2560     10.6        6                  6
        2561      9.5        5                  4
        2562      9.6        6                  7
        2563      9.0        6                  6
        2564     10.2        6                  4
        2565      9.2        5                  5
        2566      9.4        5                  5
        2567     11.7        5                  6
```

It looks like some wines have the same predicted and actual quality while some others don't. We will need to do some more work to see how our algorithm performs.

```
In [51]: import matplotlib.pyplot as plt

         fig = plt.figure()
         fig.set_size_inches(18.5, 10.5)
         ax1 = fig.add_subplot(111)

         ax1.scatter(df_testing.index, df_testing['predicted quality'],
                     s=10, c='b', marker="s", label='predicted quality')
         ax1.scatter(df_testing.index, df_testing['quality'], s=10, c='r',
                     marker="o", label='real quality')
         plt.legend(loc='upper left')
         plt.show()
```



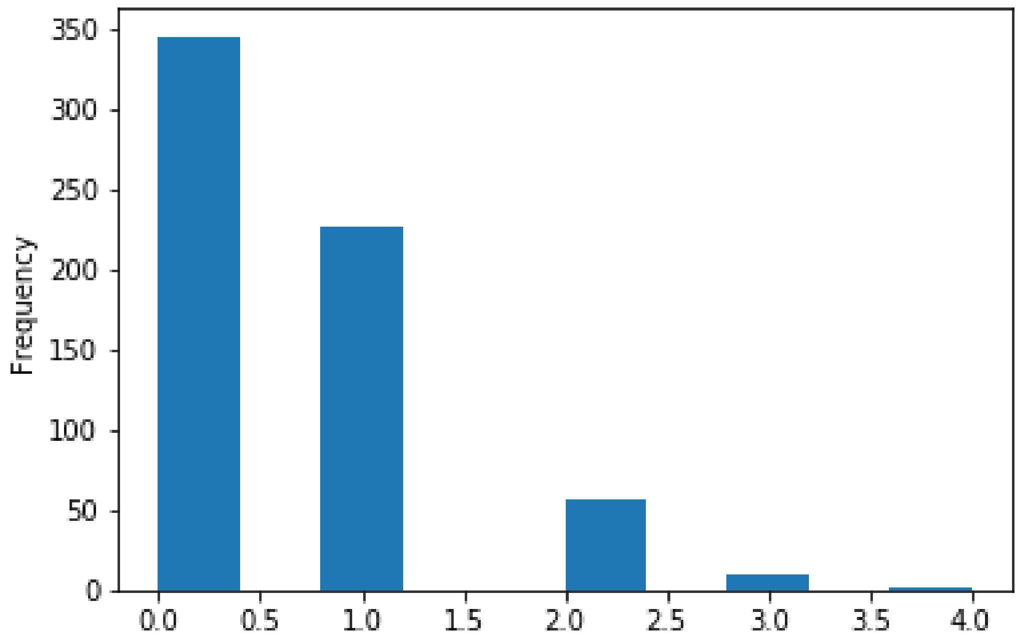
The scatter plot above does not look so nice and we cannot really tell much about our prediction by looking at it. Thus, we will calculate the root mean sqaure error we got and store the value in our dataframe.

```
In [52]: #takes in two values and calculautes their root mean square difference
         #and then returns that result
         def rmse(pred, tar):
             diff = pred - tar
             diff_sq = diff ** 2
             mean_diff_sq = diff_sq.mean()
             rmse_val = np.sqrt(mean_diff_sq)
             return rmse_val*1.0
```

```
In [53]: df_testing['rms']=0.0
         for i in df_testing.index:
                 df_testing.at[i,'rms'] = rmse(df_testing.at[i,'predicted quality'],
                                               df_testing.at[i,'quality'])
```

```
/Users/aitinghsieh/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1: SettingWithC
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.htm
  """Entry point for launching an IPython kernel.
```

```
In [54]: df_testing[:10]

Out[54]:       fixed.acidity  volatile.acidity  citric.acid  residual.sugar  chlorides  \
         2558            7.9              0.54         0.34            2.50      0.076
         2559            7.3              0.30         0.34            2.70      0.044
         2560            6.7              0.46         0.24            1.70      0.077
         2561            6.7              0.26         0.26            4.10      0.073
         2562            7.4              0.19         0.30            1.40      0.057
         2563            5.7              0.22         0.22           16.65      0.044
         2564           12.3              0.27         0.49            3.10      0.079
         2565            7.3              0.91         0.10            1.80      0.074
         2566            9.3              0.40         0.49            2.50      0.085
         2567            6.1              0.20         0.40            1.90      0.028

               free.sulfur.dioxide  total.sulfur.dioxide  density    pH  sulphates  \
         2558                  8.0                  17.0  0.99235  3.20       0.72
         2559                 34.0                 108.0  0.99105  3.36       0.53
         2560                 18.0                  34.0  0.99480  3.39       0.60
         2561                 36.0                 202.0  0.99560  3.30       0.67
         2562                 33.0                 135.0  0.99300  3.12       0.50
         2563                 39.0                 110.0  0.99855  3.24       0.48
         2564                 28.0                  46.0  0.99930  3.20       0.80
         2565                 20.0                  56.0  0.99672  3.35       0.56
         2566                 38.0                 142.0  0.99780  3.22       0.55
         2567                 32.0                 138.0  0.99140  3.26       0.72

               alcohol  quality  predicted quality  rms
         2558     13.1        8                  6  2.0
         2559     12.8        8                  6  2.0
         2560     10.6        6                  6  0.0
         2561      9.5        5                  4  1.0
         2562      9.6        6                  7  1.0
         2563      9.0        6                  6  0.0
         2564     10.2        6                  4  2.0
         2565      9.2        5                  5  0.0
         2566      9.4        5                  5  0.0
         2567     11.7        5                  6  1.0

In [55]: df_testing['rms'].plot.hist()

Out[55]: <matplotlib.axes._subplots.AxesSubplot at 0x1a0d91fc18>
```



We can see that the error we get is actually very small. The predicted quality is in the most case actual quality ś 1, with a few ones having a more inaccurate result. So after all, our prediction is still not bad.

8 Part 5 - Ethics and Privacy

In our ventures in trying to determine what predicts wine quality, it is important to consider who and what may be negatively impacted by the results we obtain. Our results may reveal that certain qualities in wine are more favorable than others, affecting the business of companies that may be known to have more or less of a favorable quality. For example, if it is found that a wine with a higher sugar content is less favorable, companies and vineyards known for sweeter wines may find their businesses negatively impacted. Additionally, if a certain company claims the key to quality is one specific physicochemical property and we find their information to be false, that company may also find their business to be negatively impacted. With more awareness as to what makes a quality wine, perceived high quality wine price could drop and perceived low quality price could increase, if our results dispute their perception.

Additionally, it is important to consider that our data is limited, biased, and skewed. We cannot be sure that the data was collected in a way that is accurate, precise, and equitable. Since all our data is from one wine company, we are sure that all variations of wine in the world and that all physicochemical properties affecting taste are not represented in our dataset. With this in mind, if our results grow in popularity, it can lead to positive or negative business for the company who provided this data, depending on results. Lastly, we know that wine quality is subjective and the “rating” portion of our data is biased. With biased ratings, it is also likely that the results are skewed, with more ratings at extreme ends than close to a median. With all this in mind, we also consider that our results cannot be fully used to defend a hypothesis, and that outside evidence is necessary for full confirmation.

9 Part 6 - Conclusion & Discussion

In conclusion, we found that though we are able to somewhat predict wine quality, it is still not fine tuned enough to use ethically. Our prediction is generally on par, sometimes slightly off, than the real quality, within 2 at most. In relation to our hypothesis, all our factors of physicochemical features combine to create different optimal wines, meaning that all of our factors are involved in creating the proper flavor profile. Our original hypothesis predicts that lower volatile acidity, medium alcohol percentage, and higher citric acid would create the best quality, and we did not necessarily find that to be true. Interestingly, though we are already working with 12 variables, there are still contributors to wine quality that are missing. With each of the 12, we were unable to find one that stood out or a few that stood out together in making a quality wine. They all needed to combine with one another in a grand way to make a high quality wine.

A possible confound is in variables that cannot directly be measured from the wine itself. Many experts cite the climate and soil at the vineyard in which the wine originated, something that was not taken into account with our data. This may account for part of the inaccuracy we faced. Limitations in our data may also account for inaccuracy. As stated before, we cannot be sure of the precision with which the data was collected. Additionally, all of the wine in our dataset came from the same vineyard, limiting our scope. Our dataset also lacked wine at the extreme ends of high and low quality, limiting our training, thus leading to inaccuracy. A few confounds are inevitable, but it may have done us well to try to alleviate some of their effects.

If we were to continue this project, it may help to look deeper in to advanced data science techniques for tackling problems like this. In our pursuit to predict what makes a high quality wine, we found and learned about KD Tree, which made sense of our data when we could not. There may be more techniques out there that would've better our predictions and help us properly pinpoint the best wine. Additionally, the addition of more data may have helped us better find the correlations we were looking for. Despite our attempts, the one true perfect wine is still somewhat of a mystery, but we are sure data scientists will tackle it better than we could.