Exploratory Data Analysis and Classification Algorithm Testing on Red Wine Quality Data

Kaan Dincer, Bryan Coronel, David Armstrong
Washington University in St. Louis

ESE 417: Introduction to Machine Learning & Pattern Classification

Dr. Jinsong Zhang

May 6th, 2022

Introduction

There are countless nuggets of gold hidden within datasets that common machine learning techniques can uncover. Routine and straightforward data collection procedures can hold unknown correlations and interdependencies that could serve as useful predictors for desirable features. In the case of red wine quality data, simple physicochemical tests likely ran during production contain actionable predictors for what the quality of the wine would be rated in future.

Throughout this report, data cleaning, dimension reduction, classification, and hypertuning processes are implemented in order to find the best features to predict the rated quality of wine. Simple checking for erroneous or null data and column name formatting is performed on the almost completely cleaned dataset. Principal component analysis is employed to optimize performance of the models and make them more simple to avoid overfitting. Results are visualized to determine its viability. K-nearest neighbors and support vector machine algorithms from the Scitkit-learn package are utilized to find how clusterizable the different quality ratings are. Finally, once the best models are chosen from each algorithm, they are run through hypertuning functions to choose the best value for "K" and kernel function, respectively. In the end, a multitude of performance indices and visual indicators are used to comprehensively determine accuracy.

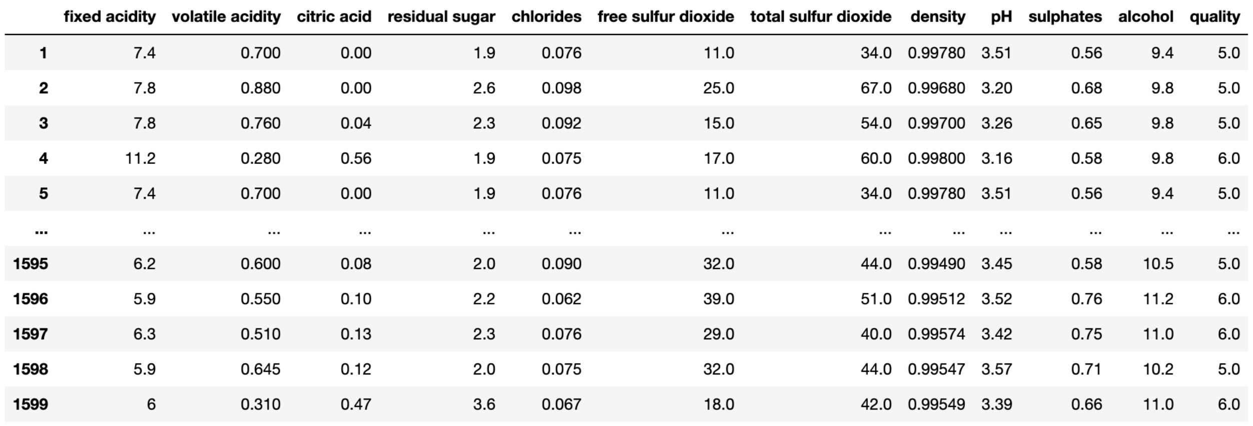
A brief description of the data includes physiochemical test performed on red wine collated with the eventual rating of the quality. The dataset appears to have been cleaned and was packaged by the University of California, Irvine.

It is easy to see to the potential impact and refinement for future iterations of the product via the techniques mentioned above. Machine learning, specifically classification algorithms, can be powerful tools for the prediction of business outcomes.

In the end, results were quite average and a bit far from what would be deployed at a production-level. Further improvement of the accuracy would require more sophisticated models, more data, more features, or all of the above.

Methods

As mentioned earlier, data cleaning, dimension reduction, classification, and hypertuning processes are employed throughout the analysis. To go into more detail, the data cleaning process required implementing a new header manually. Using Pandas to read in the data, the first row was discarded as it was a row containing column names that couldn't be formatted properly. Additionally, the data was coerced from string to numeric data type. This allowed for the checking and removal of null values. Continuing the data exploration, histograms were generated of all of the columns to find any visual patterns immediately. Once the data has been properly cleaned and explored, more preprocessing tests can be performed.



	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
1	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5.0
2	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5.0
3	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5.0
4	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6.0
5	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5.0
...	...	...	...	...	...	...	...	...	...	...	...	...
1595	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5.0
1596	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	6.0
1597	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6.0
1598	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5.0
1599	6	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6.0

Figure 1: Snapshot of dataframe showing each attributes and a small sample of values

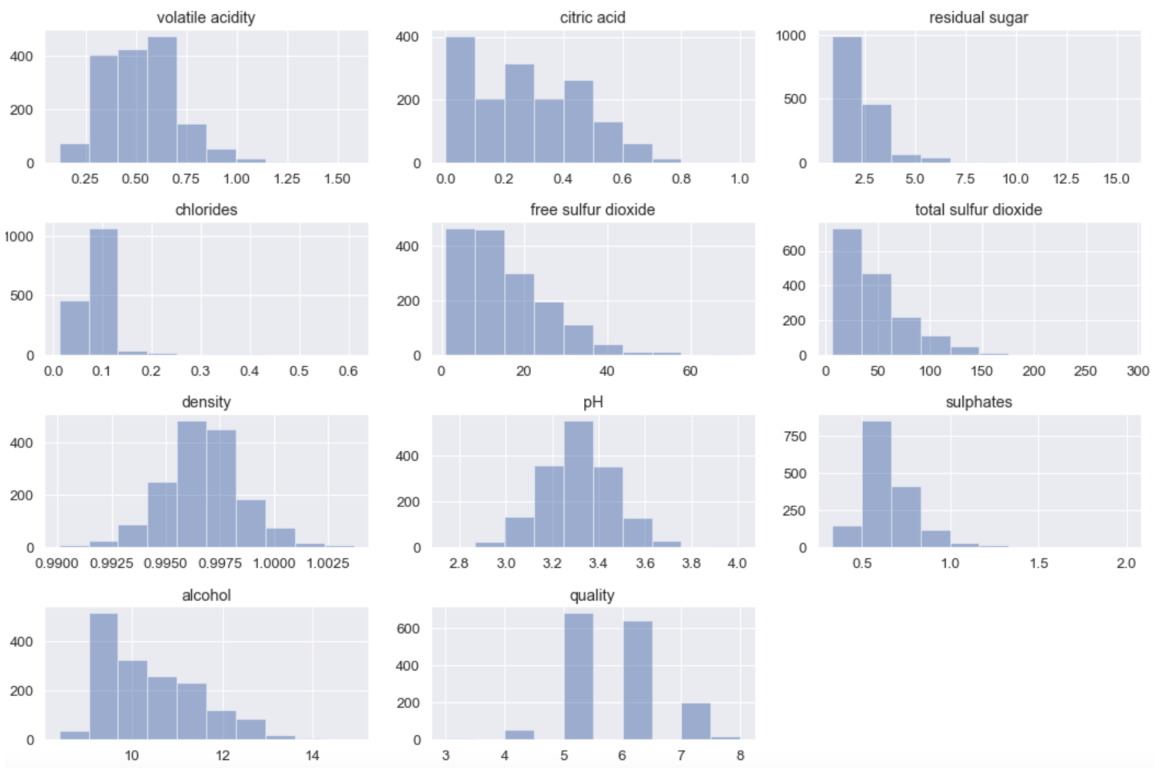


Figure 2: Histograms of each data attribute where we can see quality is usually 5 or 6

A common technique used before most classification algorithms to simplify a model is principal component analysis. The main idea behind this method is that it transforms high-dimensions data into lower-dimensions while retaining as much information as possible. PCA is extremely useful when working with data sets that have a lot of features. While having more data is always great, occasionally some of it is unnecessary and can be condensed. In practice, we would have impossibly long training time and as discussed in class, the curse of dimensionality starts to become a problem. After a simple run of the model, using two components, the clusters found overlapped too much for dimension reduction to be a viable modification to the data. Therefore, the original data was used for training in the subsequent classification algorithms.

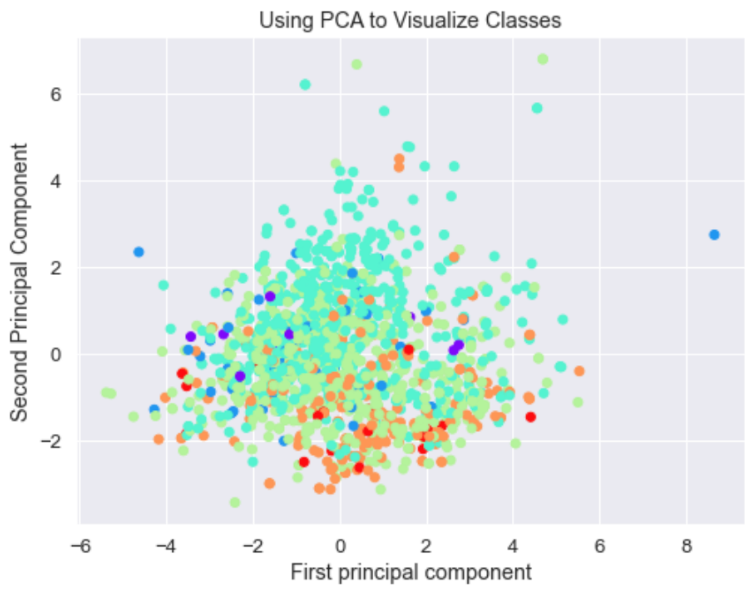


Figure 3: Overlapping Principal Component Analysis

Firstly, Scikit-Learn's K-Nearest Neighbors algorithm was implemented from values of K = 3 and 9. Accuracy scores were calculated for each value of K, and each choice was outputted and visualized. Additionally, a Scitkit's metrics class was utilized to score the model across a variety of methods including precision, recall, f1-score, and support. Making use of classification_report(), these scores are calculated for every class present in the target variable and provide an average for a detailed summary of where the model fails to classify data. Similarly, a Support Vector Machine was then used to see if an improvement could be made. Like with the KNN algorithm, SVM has a few options for how the algorithm can produce vastly different results. For SVM, this is dependent on the kernel functions used. Consequently, a loop was used to run through the different kernel functions Scikit-Learn's implementation of SVM is able to take in and calculate their scores. The kernel function with the highest score from Scikit-Learn's accuracy_score() function was used moving forward to tune hyperparemeters.

Support Vector Machines lend themselves well to trying out different combinations of parameters to maximize accuracy. This is the exact purpose of classes from Scikit-Learn such as model_selection and its function GridSearchCV(). Although it takes a while, the function will select a model from hyperparameters that are open to tuning. In the case of this model, parameters "gamma" and "C" were tweaked around. In the end, both classification models with their calculated hyperparameters will be used to generate predictions and visualizations will be made to assess their performance.

Results and Analysis

<u>KNN Method</u>

For the KNN method, models were trained based on 3 and 9 nearest neighbors. The accuracy scores shown below are 46.9% for K=3 and 51.9% for K=9, making 9 the better choice of parameter value for this dataset. The confusion matrix below shows that the 2nd choice of y-values are often confused with the 3rd choice of y-values which corresponds to Quality values of 5 and 6. With multi-label classification, our "Accuracy Score" is a harsh subset accuracy, giving the proportion of correct labels in our test set.

```
Accuracy Score (K=3):
 0.46875
```

```
Accuracy Score (K=9):
 0.51875
```

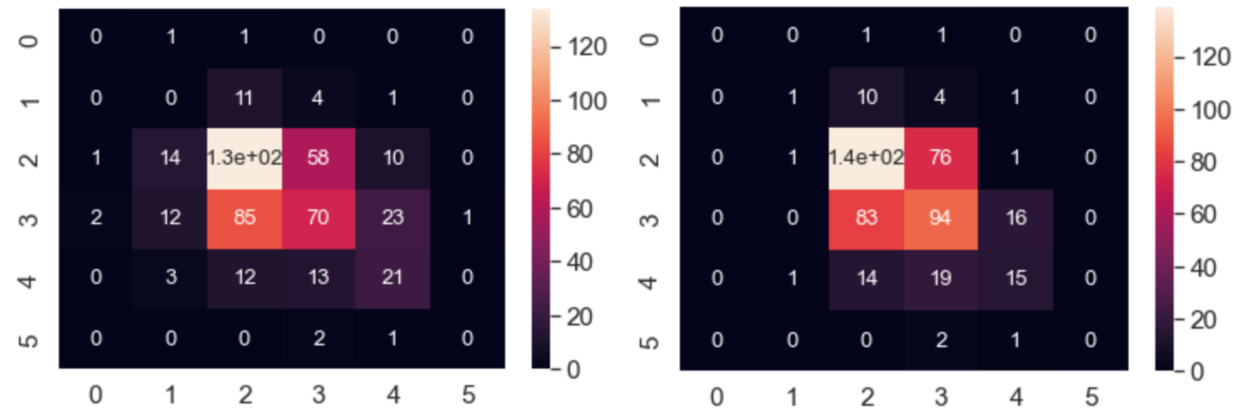


Figure 4: Confusion matrices for KNN where K = 3 and 9

```
              precision    recall  f1-score   support

         3.0       0.00      0.00      0.00         2
         4.0       0.00      0.00      0.00        16
         5.0       0.55      0.62      0.58       217
         6.0       0.48      0.36      0.41       193
         7.0       0.38      0.43      0.40        49
         8.0       0.00      0.00      0.00         3

    accuracy                           0.47       480
   macro avg       0.23      0.23      0.23       480
weighted avg       0.48      0.47      0.47       480
```

```
              precision    recall  f1-score   support

         3.0       0.00      0.00      0.00         2
         4.0       0.33      0.06      0.11        16
         5.0       0.56      0.64      0.60       217
         6.0       0.48      0.49      0.48       193
         7.0       0.44      0.31      0.36        49
         8.0       0.00      0.00      0.00         3

    accuracy                           0.52       480
   macro avg       0.30      0.25      0.26       480
weighted avg       0.50      0.52      0.51       480
```

Figure 5: Classification reports for KNN where K = 3 and 9, respectively

<u>SVM Method</u>

For the SVM Method tuning the hyperparameters gave us the model to be linear, gamma to be 0.56 and C to be 1 with a grid score of 0.55. Running the Support Vector Machine method using these parameters gave us an accuracy of 62.3%, which showed us our model did not have great accuracy. The model's mean squared error was 0.458.

The confusion matrix in Figure 3 shows that the 2nd choice of y-values are often confused with the 3rd choice of y-values which corresponds to Quality values of 5 and 6. We also see that the 3rd choice of y values are often confused with the 2nd and 4th choice of y values corresponding to quality values 5 and 7.

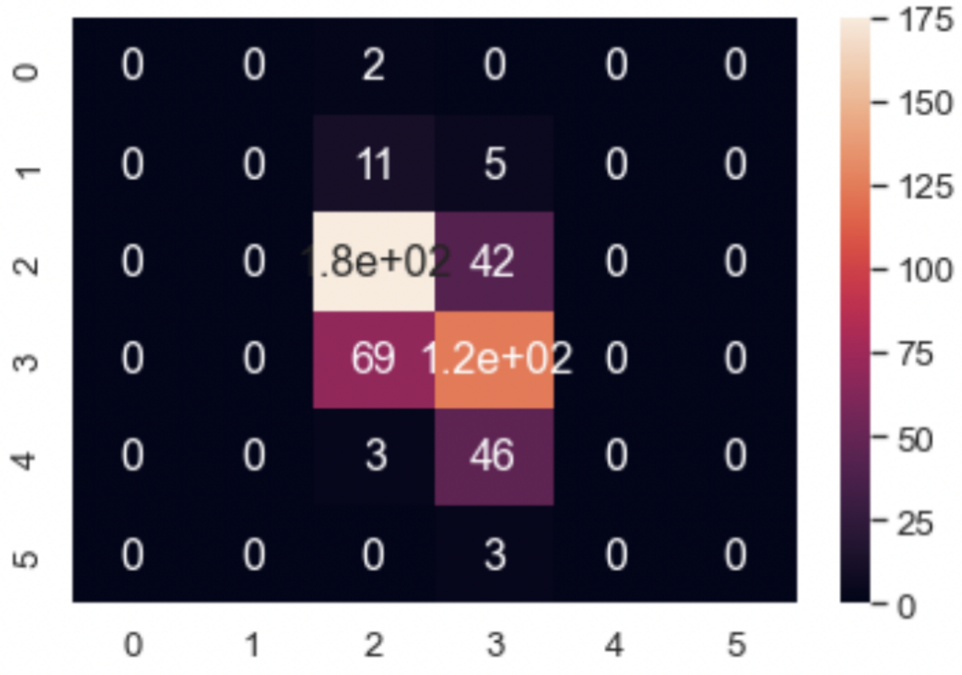
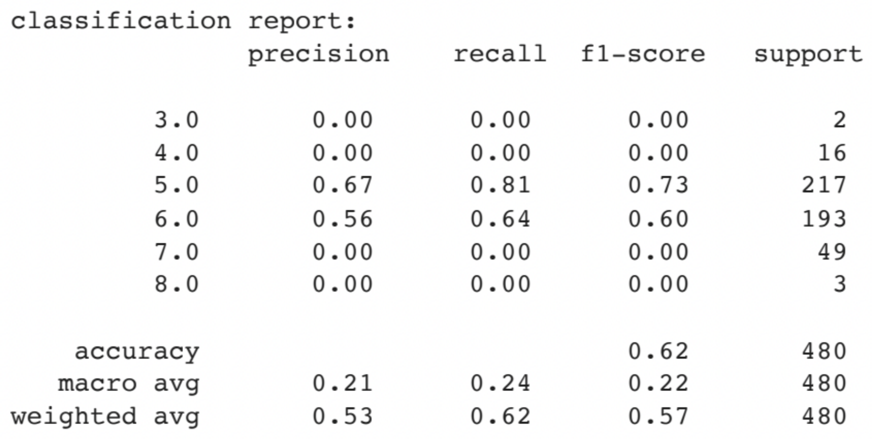


Figure 6: Confusion matrix for the SVM Method

The classification report resulting from our method is given below, in Figure 4.



```
classification report:
              precision    recall  f1-score   support

         3.0       0.00      0.00      0.00         2
         4.0       0.00      0.00      0.00        16
         5.0       0.67      0.81      0.73       217
         6.0       0.56      0.64      0.60       193
         7.0       0.00      0.00      0.00        49
         8.0       0.00      0.00      0.00         3

    accuracy                           0.62       480
   macro avg       0.21      0.24      0.22       480
weighted avg       0.53      0.62      0.57       480
```

Figure 7: Classification Report from the SVM Method

Conclusions

To summarize, the best KNN method yielded an accuracy score of 51.9% while the best SVM method yielded an accuracy score of 62.3%. Therefore, we determine that using SVM method with a linear model and gamma of 0.56 is the optimal choice out of the methods tried.

Appendix

In [136…

```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn import svm
from sklearn.datasets import make_blobs
from sklearn import metrics
from sklearn.svm import SVC
from sklearn.decomposition import PCA
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

In [137…

```python
# download & read data

headerList = ['fixed acidity' ,'volatile acidity', 'citric acid',"residual sugar
data = pd.read_csv('winequality-red-1.csv', sep=";",header=None, names = headerL
data = data[1:]
#Creating X and y
X = data.loc[:, 'fixed acidity':'alcohol']
y = data['quality']

data
```

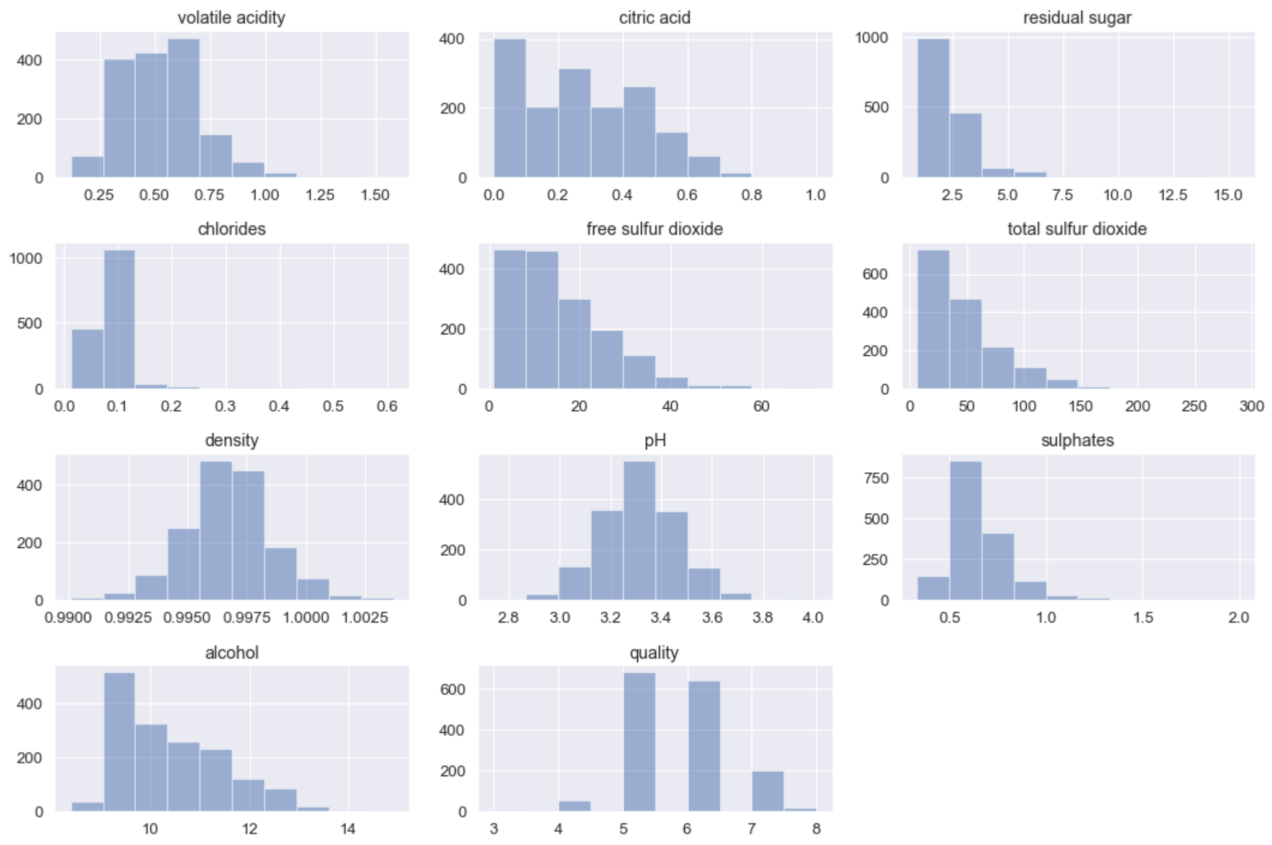
Out[137…

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alco
1	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	
2	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	
3	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	
4	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	
5	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	
...	...	...	...	...	...	...	...	...	...	...	
1595	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	
1596	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	
1597	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	
1598	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	
1599	6	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	

1599 rows × 12 columns

In [138…

```
data.hist(alpha=0.5, figsize=(15, 10))
plt.tight_layout()
plt.show()
```



In [139…

```
for h in headerList:
  data[h] = pd.to_numeric(data[h], errors='coerce')

print("\nChecking for null values: \n")
data.isna().sum()

#We see there are no null values so no need to correct data
```

```
Checking for null values: 
```

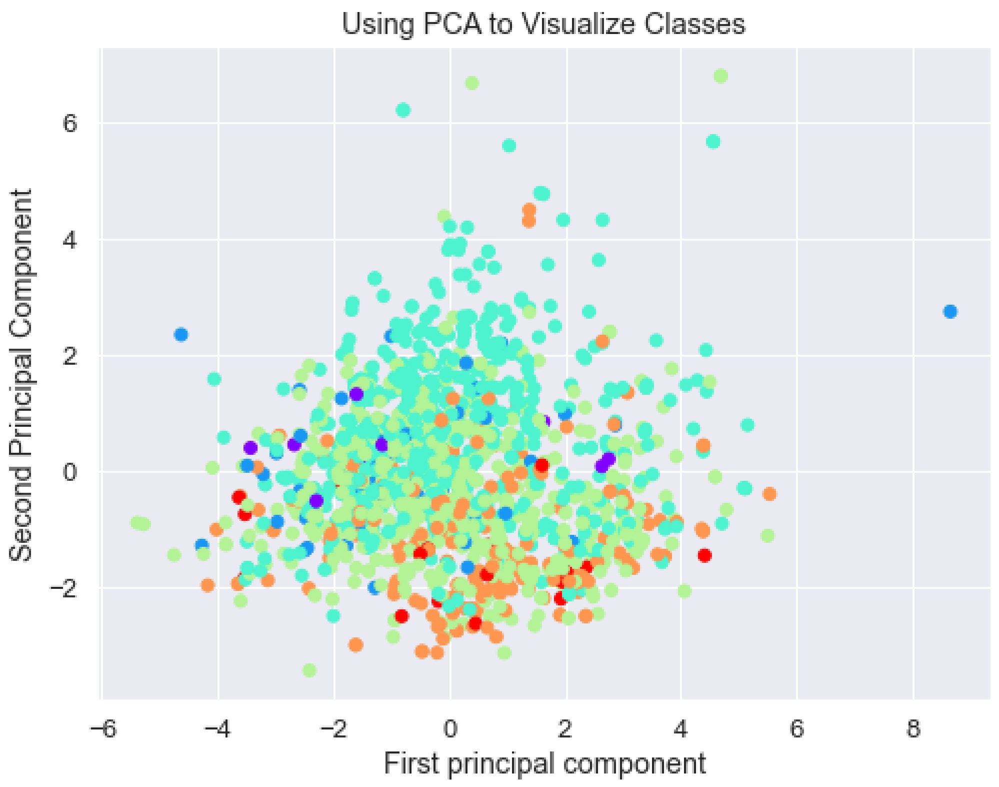
Out[139…

```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

In [142…

```
df_pca = data.copy()
X_pca = df_pca.loc[:, 'fixed acidity':'alcohol']
y_pca = df_pca['quality']
```

```
X_pca.tail()

#Preprocessing in preparation for PCA: Standardizing the predictor variables
X_pca = StandardScaler().fit_transform(X_pca)

#Fit PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_pca)

X_pca.shape

plt.figure(figsize=(8,6))
plt.scatter(X_pca[:,0],X_pca[:,1],c=y_pca,cmap='rainbow')
plt.xlabel('First principal component')
plt.ylabel('Second Principal Component')
plt.title("Using PCA to Visualize Classes")
plt.show()

# We see that even using PCA, this is a non linearly seperable problem.
```

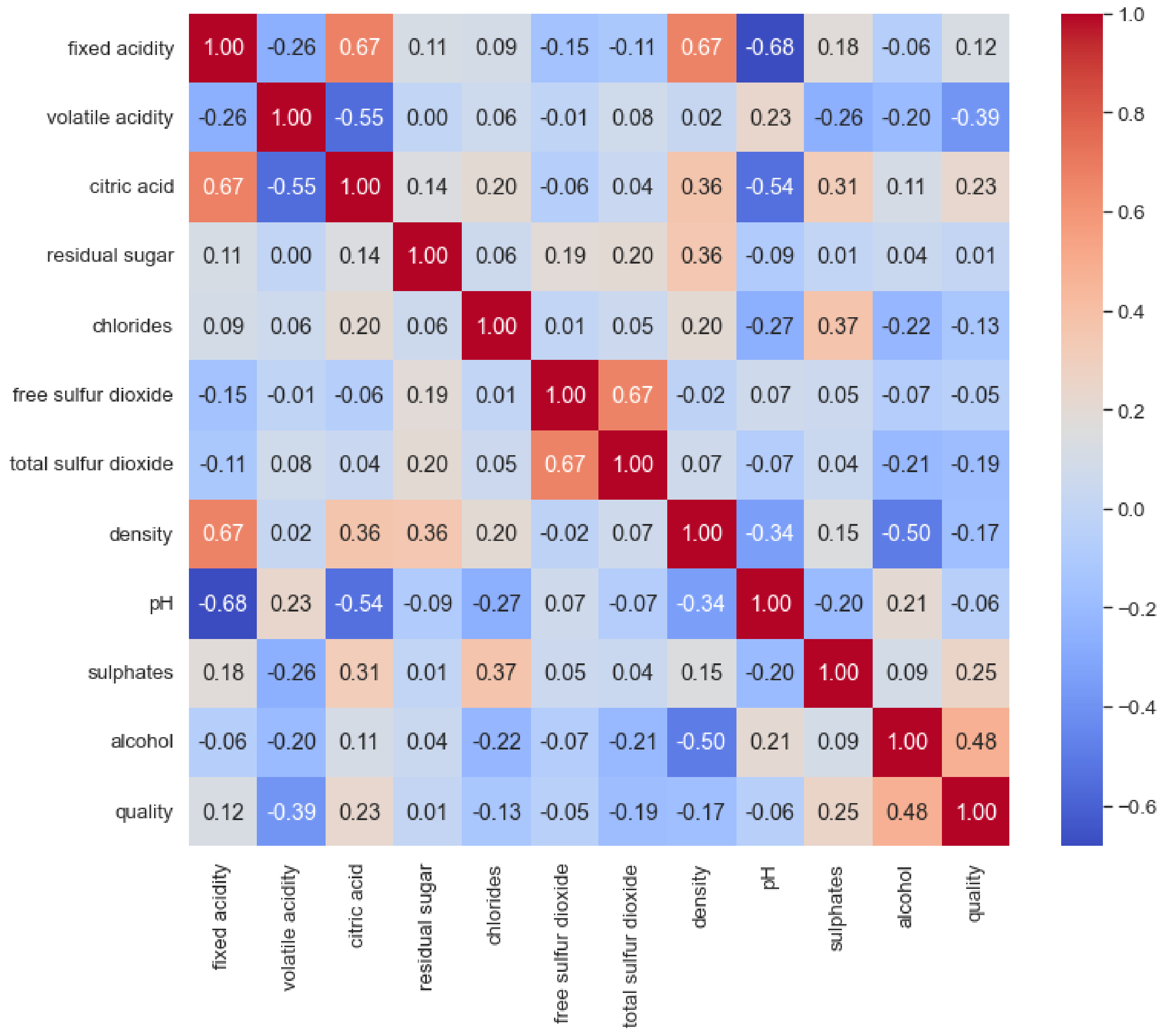


In [143…

```
import seaborn as sns

plt.figure(figsize=(12,10))
sns.heatmap(data.corr(),annot=True, cmap='coolwarm',fmt='.2f')
```

Out[143…

```
<AxesSubplot:>
```



	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
fixed acidity	1.00	-0.26	0.67	0.11	0.09	-0.15	-0.11	0.67	-0.68	0.18	-0.06	0.12
volatile acidity	-0.26	1.00	-0.55	0.00	0.06	-0.01	0.08	0.02	0.23	-0.26	-0.20	-0.39
citric acid	0.67	-0.55	1.00	0.14	0.20	-0.06	0.04	0.36	-0.54	0.31	0.11	0.23
residual sugar	0.11	0.00	0.14	1.00	0.06	0.19	0.20	0.36	-0.09	0.01	0.04	0.01
chlorides	0.09	0.06	0.20	0.06	1.00	0.01	0.05	0.20	-0.27	0.37	-0.22	-0.13
free sulfur dioxide	-0.15	-0.01	-0.06	0.19	0.01	1.00	0.67	-0.02	0.07	0.05	-0.07	-0.05
total sulfur dioxide	-0.11	0.08	0.04	0.20	0.05	0.67	1.00	0.07	-0.07	0.04	-0.21	-0.19
density	0.67	0.02	0.36	0.36	0.20	-0.02	0.07	1.00	-0.34	0.15	-0.50	-0.17
pH	-0.68	0.23	-0.54	-0.09	-0.27	0.07	-0.07	-0.34	1.00	-0.20	0.21	-0.06
sulphates	0.18	-0.26	0.31	0.01	0.37	0.05	0.04	0.15	-0.20	1.00	0.09	0.25
alcohol	-0.06	-0.20	0.11	0.04	-0.22	-0.07	-0.21	-0.50	0.21	0.09	1.00	0.48
quality	0.12	-0.39	0.23	0.01	-0.13	-0.05	-0.19	-0.17	-0.06	0.25	0.48	1.00

In [144…

```
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.3,random_s
```

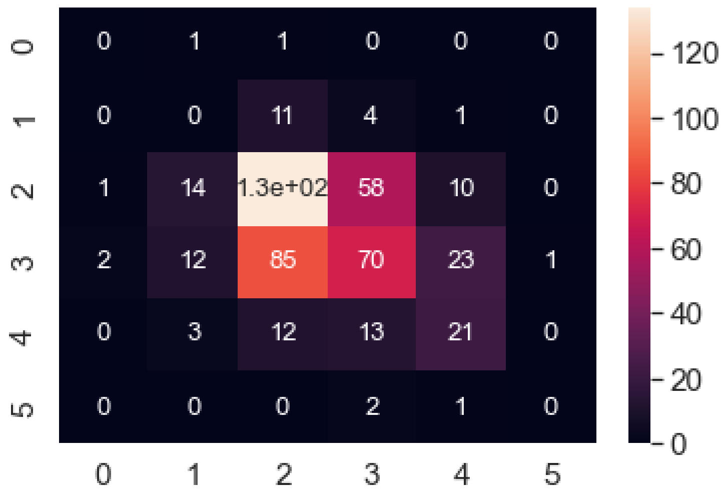
KNN Classifier

In [124…

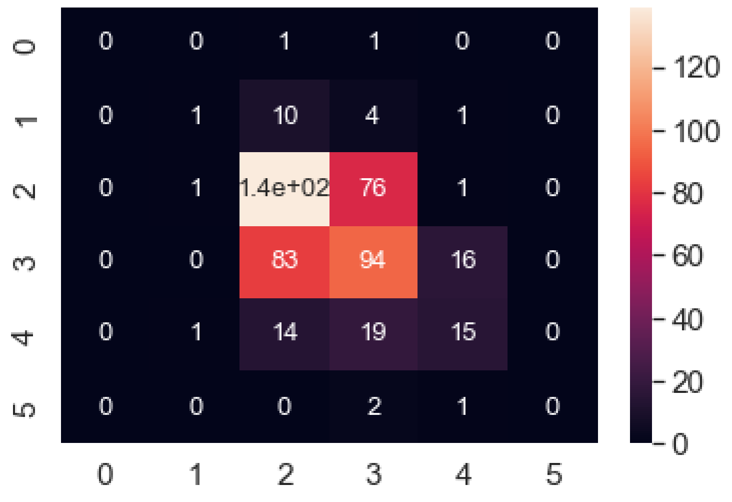
```
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X_train, Y_train)
y_pred = neigh.predict(X_test)
score = neigh.score(X_test,Y_test)
print('Accuracy Score (K=3): \n',score)

#Plot confusion matrix
cm = confusion_matrix(Y_test, y_pred)
df_cm = pd.DataFrame(cm)
sns.set(font_scale=1.4) # for label size
sns.heatmap(df_cm, annot=True, annot_kws={"size": 13}) # font size
plt.show()

#Print classification report
clas = classification_report(Y_test, y_pred)
print(clas)
```

```
#Test Different Hyperparameter
neigh = KNeighborsClassifier(n_neighbors=9)
neigh.fit(X_train, Y_train)
y_pred = neigh.predict(X_test)
score = neigh.score(X_test,Y_test)
print('Accuracy Score (K=9): \n',score)

#Plot confusion matrix
cm = confusion_matrix(Y_test, y_pred)
df_cm = pd.DataFrame(cm)
sns.set(font_scale=1.4) # for label size
sns.heatmap(df_cm, annot=True, annot_kws={"size": 13}) # font size
plt.show()

#Print classification report
clas = classification_report(Y_test, y_pred)
print(clas)
```

```
Accuracy Score (K=3): 
 0.46875
```



```
              precision    recall  f1-score   support

         3.0       0.00      0.00      0.00         2
         4.0       0.00      0.00      0.00        16
         5.0       0.55      0.62      0.58       217
         6.0       0.48      0.36      0.41       193
         7.0       0.38      0.43      0.40        49
         8.0       0.00      0.00      0.00         3

    accuracy                           0.47       480
   macro avg       0.23      0.23      0.23       480
weighted avg       0.48      0.47      0.47       480

Accuracy Score (K=9): 
 0.51875
```



```
              precision    recall  f1-score   support

         3.0       0.00      0.00      0.00         2
         4.0       0.33      0.06      0.11        16
         5.0       0.56      0.64      0.60       217
         6.0       0.48      0.49      0.48       193
         7.0       0.44      0.31      0.36        49
         8.0       0.00      0.00      0.00         3

    accuracy                           0.52       480
   macro avg       0.30      0.25      0.26       480
weighted avg       0.50      0.52      0.51       480

/Users/davidarmstrong/opt/anaconda3/lib/python3.9/site-packages/sklearn/metrics/
_classification.py:1248: UndefinedMetricWarning: Precision and F-score are ill-d
efined and being set to 0.0 in labels with no predicted samples. Use `zero_divis
ion` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/davidarmstrong/opt/anaconda3/lib/python3.9/site-packages/sklearn/metrics/
_classification.py:1248: UndefinedMetricWarning: Precision and F-score are ill-d
efined and being set to 0.0 in labels with no predicted samples. Use `zero_divis
ion` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/davidarmstrong/opt/anaconda3/lib/python3.9/site-packages/sklearn/metrics/
_classification.py:1248: UndefinedMetricWarning: Precision and F-score are ill-d
efined and being set to 0.0 in labels with no predicted samples. Use `zero_divis
ion` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [125…

```
#choosing kernel
for kernel in ('linear', 'poly', 'rbf', 'sigmoid'):
    #Create an svm model
    model = svm.SVC(kernel=kernel, C=10)

    #Train the model using the training sets
    clf = model.fit(X_train, Y_train)

    #Predict the response for test dataset
    y_pred = clf.predict(X_test)
    print(kernel,"Accuracy:",metrics.accuracy_score(Y_test, y_pred))
```

```
linear Accuracy: 0.6291666666666667
poly Accuracy: 0.5375
```

```
rbf Accuracy: 0.6104166666666667
sigmoid Accuracy: 0.31875
```

In [126…

```
# tuning hyperparameters
gamma_range = np.logspace(-4,1,5)
C_range = np.linspace(1,10,10)
param_grid = dict(gamma = gamma_range,C = C_range)

grid = GridSearchCV(svm.SVC(kernel = "rbf", cache_size=5000), param_grid=param_g
grid.fit(X_train, Y_train)
print("The best parameters are %s \nwith a score of %0.5f" % (grid.best_params_,

grid.score(X_test,Y_test)
```

```
/Users/davidarmstrong/opt/anaconda3/lib/python3.9/site-packages/sklearn/model_se
lection/_split.py:666: UserWarning: The least populated class in y has only 8 me
mbers, which is less than n_splits=10.
  warnings.warn(("The least populated class in y has only %d"
The best parameters are {'C': 1.0, 'gamma': 0.5623413251903491}
with a score of 0.54961
```

Out[126…

```
0.6020833333333333
```

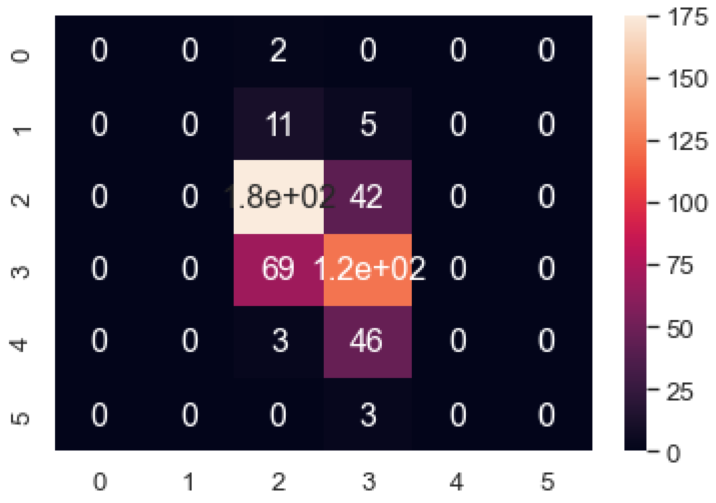
In [129…

```
# Best model
model = svm.SVC(kernel='linear', gamma = 0.5623413251903491, C=1)

#Train the model using the training sets
clf = model.fit(X_train, Y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)
print("Accuracy:",metrics.accuracy_score(Y_test, y_pred))

y = clf.predict(X_test)
print('Classifier MSE: ',mean_squared_error(y,Y_test))
```

```
Accuracy: 0.6229166666666667
Classifier MSE:  0.4583333333333333
```

In [128…

```
#Plot confusion matrix
cm = confusion_matrix(Y_test, y_pred)
df_cm = pd.DataFrame(cm)
sns.set(font_scale=1.2) # for label size
sns.heatmap(df_cm, annot=True, annot_kws={"size": 16}) # font size
plt.show()

#Print classification report
classification = classification_report(Y_test, y_pred)
print("classification report: \n",classification)
```



```
classification report:
              precision    recall  f1-score   support

         3.0       0.00      0.00      0.00         2
         4.0       0.00      0.00      0.00        16
         5.0       0.67      0.81      0.73       217
         6.0       0.56      0.64      0.60       193
         7.0       0.00      0.00      0.00        49
         8.0       0.00      0.00      0.00         3

    accuracy                           0.62       480
   macro avg       0.21      0.24      0.22       480
weighted avg       0.53      0.62      0.57       480

/Users/davidarmstrong/opt/anaconda3/lib/python3.9/site-packages/sklearn/metrics/
_classification.py:1248: UndefinedMetricWarning: Precision and F-score are ill-d
efined and being set to 0.0 in labels with no predicted samples. Use `zero_divis
ion` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/davidarmstrong/opt/anaconda3/lib/python3.9/site-packages/sklearn/metrics/
_classification.py:1248: UndefinedMetricWarning: Precision and F-score are ill-d
efined and being set to 0.0 in labels with no predicted samples. Use `zero_divis
ion` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/davidarmstrong/opt/anaconda3/lib/python3.9/site-packages/sklearn/metrics/
_classification.py:1248: UndefinedMetricWarning: Precision and F-score are ill-d
efined and being set to 0.0 in labels with no predicted samples. Use `zero_divis
ion` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
In [ ]:
```