Using Artificial Neural Networks, K-Nearest Neighbors, and Random Forest for Classification of Red Wine Quality

ESE 417 - Introduction to Machine Learning
Final Project

Will Li - Tillman James - Matt Donaldson

December 19, 2021

1 Introduction

Optimization theory and machine learning have been studied extensively in past years and, until the recent advancement of computing power, haven't been able to do some of the complex computations within the field. Machine Learning is a subset of Artificial Intelligence and focuses on algorithms and models aimed to extract patterns out of the data. There are three main categories of Machine Learning, supervised learning, unsupervised learning, and reinforcement learning. Here, we use several supervised Machine Learning algorithms to classify the rating of a red wine.

In this project, we aim to use three different types of Machine Learning models to predict the classification of various red wines with the goal of designing, implementing and comparing the performance of each algorithm. The three models we use are an Artificial Neural Network, Random Forest, and K-Nearest Neighbors. The data set consist of 1,600 number of red wine instances and each instance contain eleven features such as fixed acidity, alcohol content, and pH. To implement this model we will use Sci-Kit Learn and several basic support packages in python to implement the various prerpossessing and machine learning models. Jupyter notebooks is where we plan to produce code and explain the various steps we took a long the way. The GitHub repository for this project can be found here here. We found that random forest model came out with the highest accuracy and precision, 70%, after we tuned the hyper-parameters of the model. Despite this, the other two models, KNN and ANN, both produced results that are not far off from the Random Forests results.

2 Methods

The data set we were working with contained 11 features of various red wines that are classified on a scale of 0-10, with 0 being bad and 10 being very good. Although this ranking is subjective, the purpose of our report is to focus on the implementation of the model and the various steps that need to be taken to accurately produce the best classifier. This project was divided into three different sections, the first being data cleaning and preprossessing, second being model implementation and training, and third hyper-parameter tuning. Each section will be detailed below and explained thoroughly.

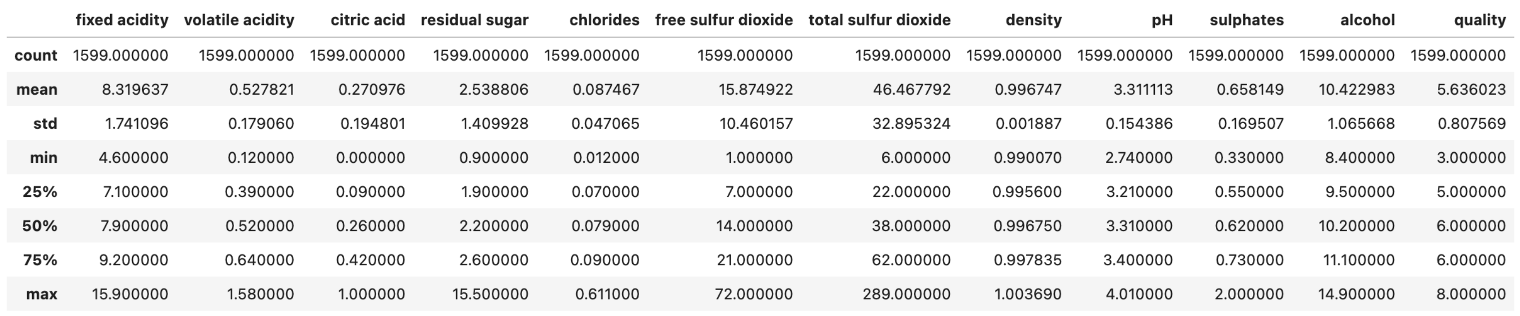
2.1 Data Cleaning and Preprocessing

Data cleaning and preprocessing consisted of manipulating the data before training or testing the machine learning model. The purpose of this is to make sure that the data we will use to train and test the model is appropriate and shaped correctly as well as give us an ideas as to what the data actually looks like in a comprehensible space. Techniques like exploratory data analysis, principle component analysis, and scaling or normalization were used to do such things. The data set contained 11 features as described in Table 1.

Features and Descriptions
Fixed Acidity: Most acids involved with wine or fixed or nonvolatile
Volatile Acidity: The amount of acetic acid in wine
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
Chlorides: The amount of salt in the wine
Free Sulfur Dioxide: Free form that exists in equilibrium btw molecular (dissolved gas) and bi-sulfite ion
Total Sulfur Dioxide: Amount of free and bound forms of
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)
Sulphates: A wine additive which can contribute to sulfur dioxide gas () levels
Alcohol: The percent alcohol content of the wine
Quality: Output variable (based on sensory data, score between 0 and 10)

Table 1. The given features of the data set with corresponding description and purpose in wine flavor.

In order to gain an understanding about the underlying characteristic of each feature in the data set we used the command "df.desribe()" from pandas to produce a statistic dataframe. This can be seen in Figure 1. Also, we have a correlation plot (in Figure 2) showing the correlation between each feature and quality. By the observation of the raw data set distribution(Figure 3), we can clearly tell that our data is very non-linearly distributed. And the correlation plot shows us the relation between each feature and how does each feature effect each other for red wine. For example, the correlation between density and alcohol is -0.5 which means that if the red wine has higher density then the alcohol index will be low. We can use all those information to clean the raw data, minimize noise and feed our model with most valuable information.



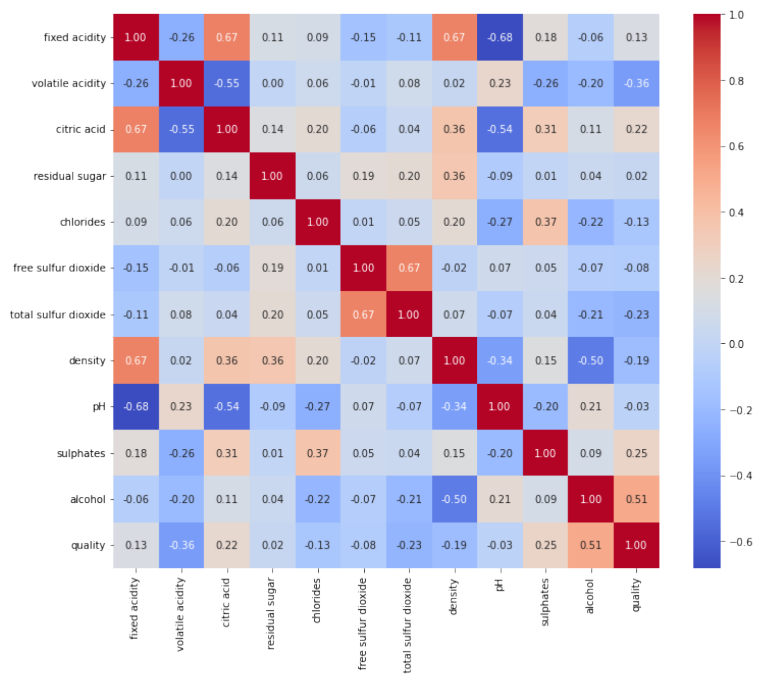
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113	0.658149	10.422983	5.636023
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0.169507	1.065668	0.807569
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000	8.400000	3.000000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.550000	9.500000	5.000000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0.620000	10.200000	6.000000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0.730000	11.100000	6.000000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.000000	14.900000	8.000000

Figure 1: Descriptive statistics of each feature within the data set.

Here we can see that the some features have very big maximum values while others have a maximum of 1. Since K-Nearest Neighbors uses Euclidean distance we can infer that this might be a problem. Thus, it suggests that we need to scale or normalize each feature in the data set. We also found that there were no null values within our data set, so there was no need to discard or impute any instances within the data set. After this we produced histograms for each feature to visualize each distribution and understand the skewness that exist for each feature. This can be seen in Figure 2.

One can see that some of the features have a fairly normal distribution, such as pH or density, while others are fairly skewed, such as total sulfur dioxide or residual sugar. This might produce some inaccuracies within our classification models.

After looking closer at the quality histogram, which represents the number of instances in each class (0-10), we see that our data set is very imbalanced. We also noticed that there are no instances that belong



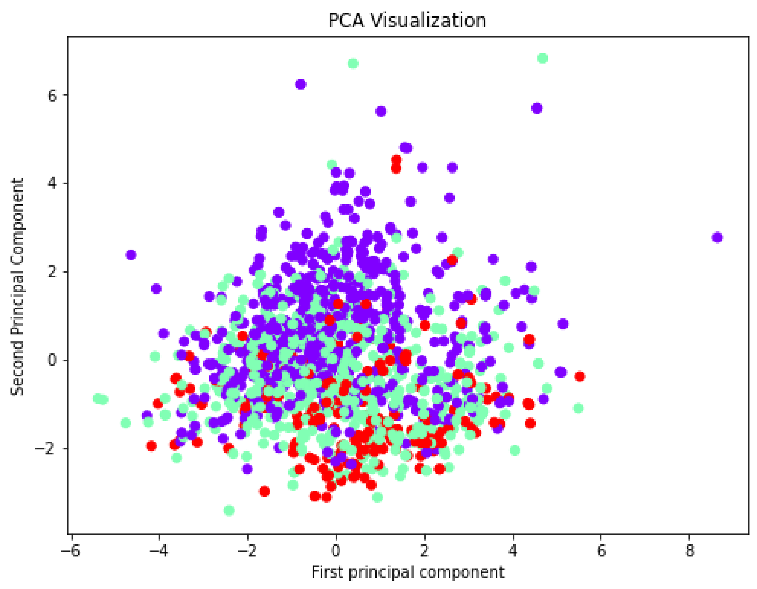


Figure 2: correlation of each feature within the data set. (left)

Figure 3: Principal component analysis (PCA) of red wine data set. (right)

to classes 0, 1, 2, 9 and 10. Classes 5, 6, and 7 account for about 95% of data within the overall data set. This means that no matter how accurate our model is, this will almost never correctly classify a wine that belongs to classes 0-4 or 8-10. To deal with this we merged the smaller groups that are present (classes 3, 4, and 8) with their "closest relative". So instances in classes 3 and 4 were compiled into class 5 and class 8 instances were combined into class 7 instances. By doing this we shortened our window of classification and only allowed us to classify wines that belong to classes 5, 6, 7. This made the data more distributed, which as a result would increase the accuracy of our models.

Next, we split the preprocessed data into training and testing. For the training data, we have 33% testing data and 67% training data. Commonly, we split training and testing data with 7:3 ratio to best feed our models with enough information and also have enough data to test their performance. In order to improve the performance of classification method, we applied StandardScaler and MinMaxScaler to our processed data. It is important to note here that the two normalization methods used were done separately. StandardScaler mainly helps us to standard normal distribution where we set mean = 0 and scale the data to unit variance. While, MinMaxScaler allows us to scale all data features in the range $[0, 1]$. By doing this, we can narrow down the range of feature values and also balance the feature scales.

2.2 Model Selection and Training

2.2.1 K-Nearest Neighbor

The first method implemented was K-Nearest Neighbor (KNN). KNN is an instance based method that rely on the fact that if two instances have like features then they would probably have the same output. So essentially the algorithm looks at the k nearest data points by calculating distance and then classifying the new data point based on the most number of representatives in a class. KNN is a clustering algorithm and does not need training since there is no learning involved. We choose KNN due to the face that is a good classification algorithm and its simplicity. In addition to this, the dimension of the data set is low, thus showing promising results from KNN. Since the dimension of the data set is low, KNN could potentially could be good. On the other hand, based on data visualization and class count, the data is not

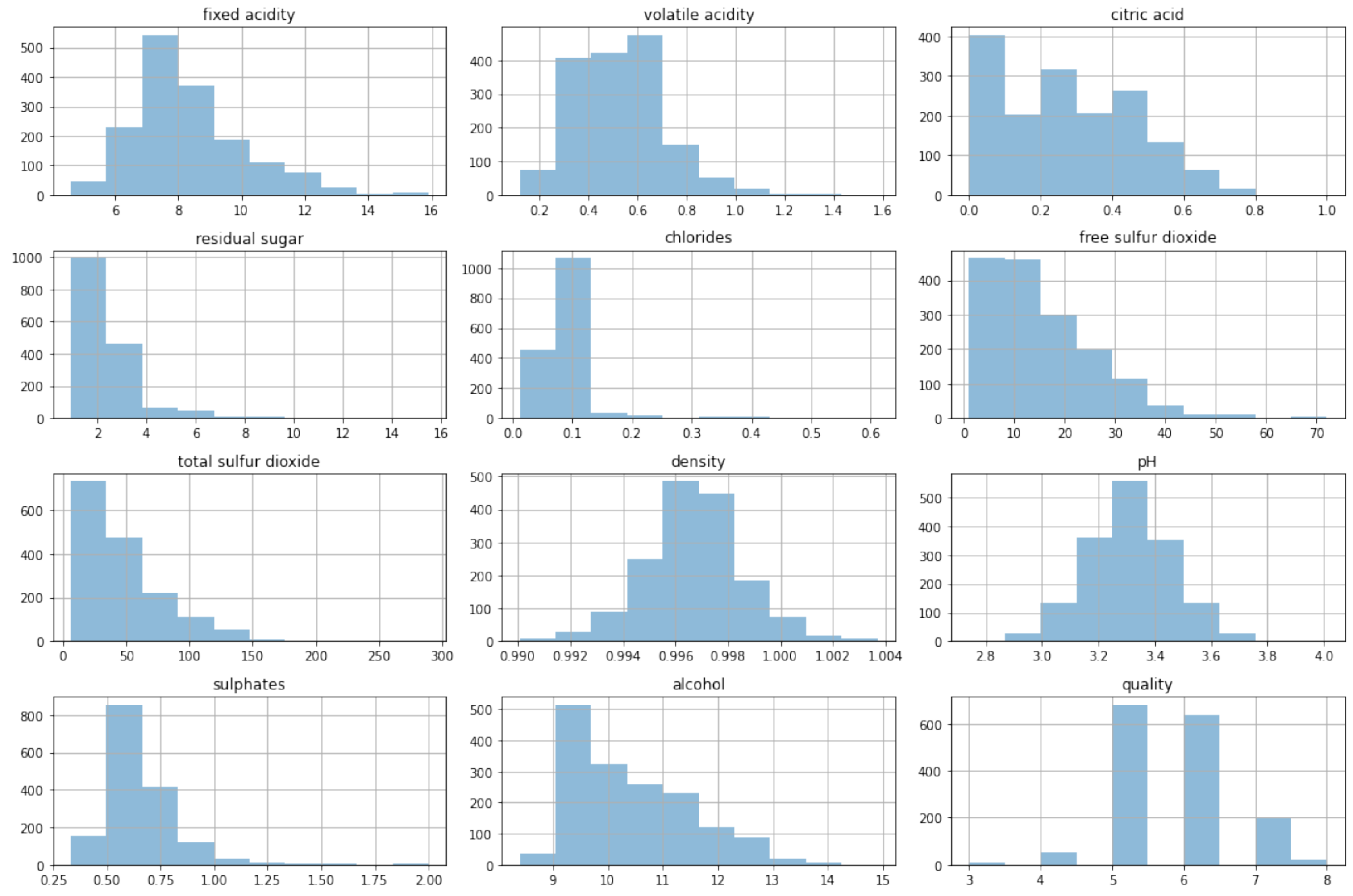


Figure 4: Histograms of each feature giving a visualization of the underlying distribution.

uniformly distributed nor linearly separable which could pose a problem. The method was implemented using Sci-Kit Learn and arbitrarily choosing a K based on the number of data points we have. This is to investigate the initial performance of the method. Once we had a good understanding of how different K parameters affected the accuracy of the model, we choose to use Grid Search to find the most optimal K that would give us the best performance (hyperparameter turning).

2.2.2 Random Forest

Lastly, Random Forest (RF) used as a final comparison. Random Forest uses hundreds of decision trees, which is called an ensemble model. To parts of RF that contribute to the algorithm is that it does bootstrap sampling of training data when building decision trees and has random subset of features to consider when splitting the nodes. So Random forest in the end takes the average of all the decision trees when predicting. There is a trade off in RF between correlation and strength. Increasing correlation will increase the error rate and increasing strength of a decision tree will lower the error rate. These two properties depend on the number of features and as you reduce the number of features, you reduce both correlation and strength. So there is a trade off.

We choose to implement a RF model due to the power of it in addition to using it as a base expectancy for accuracy. We initially built this model with 200 decision trees and a maximum depth of 100 and the max features as square root. By doing this random initialization we were able to obtain an accuracy of 72% and an F1 Score of 0.80. After finding this we used Grid Search to find the optimal number of estimators and depth of the tree.

2.2.3 Artificial Neural Network

The second method used was an Artificial Neural Network (ANN). ANN learns to model a data set through examples without making task specific rules. The basic idea behind ANN is that it takes multiple perceptron models (which are linearly separable models) and combines them in order to make a model that can handle nonlinear data sets. The structure of an ANN consists of an input layer, one or several hidden layers, and an output layer. The number of nodes within the input layer corresponds to the number of features within the data set. The number of nodes in each of the hidden layers is to be choose during cross validation but arbitrarily choose initially. The number of nodes in the output layer corresponds to the number of classes within the data set and the numerical output is the probability that the current instance being used belongs to a given class. The way ANN learns is by taking input values and computes an output from hidden layers. Then ANN compares the error between what it predicted and the actual class label of the input instance. From this the model will update its weights by back propagation and try again on a new instance within the data set.

Due to an ANN's flexibility and ability to learn very detailed patterns from complex high dimensional data sets, we choose to implement this model. In our project we chose 1 hidden layer with 700 nodes initially and used the Relu as the activation function. The number of nodes in the hidden layer was choose arbitrarily but we choose to use one hidden layer since the data is nonlinear and we do not want to overfit our model. It should be noted that the a good estimate for the total number of weights in the network is the one-tenth of the number of training instances. In this case the total number of weights should be around 150. After building and training the model with one hidden layer of 700 nodes, we used Grid Search to find the optimal number of nodes in the hidden layer with the optimal activation function.

2.3 Hyperparameter Tuning

To tune the hyperparameters of each method, GridSearchCV was used from Sci-Kit Learn. This method is an exhaustive search over all the defined parameters for a given model. For example in Random Forest, the grid search can test over hyperparameters such as number of estimators (number of decision trees in the forest) and the maximum depth of each of those decision trees. To implement GridSearchCV for each model, first the model needs to be initialized. Then the hyperparameters that need to be tuned are specified in a dictionary called the parameter grid. Once the parameter grid has been specified in the grid search, the last input needed is the cross-validation integer. The Cross Validation Integer specifies the number of folds the data will be split up into for cross validation. For KNN we tuned the K-Nearest Neighbors that the method used. For the Random Forest method we tuned the depth and number of estimators in the method. Lastly, for ANN, we tuned the activation function and number of nodes in the hidden layers. By preforming cross validation, we are able to find the most optimal hyperparameters to use when actually using our model on live data.

3 Results and Analysis

To compare the results for each method, we looked at the accuracy and the macro average precision of each method when training the model and when implementing each model with tuned hyperparameters. We first trained each model on two different sets of scaled/normalized data. The first using the MinMax Scaler and the second using the StandardScaler. The first only normalized the values between 0 and 1 while the second shifts the data so that it follows a distribution with mean 0 and standard deviation 1. After comparing these results, we used the type of data that achieved the highest accuracy score. Then implemented each model with the most optimal hyperparameters and correct data scaling technique. A table comparing the results of the accuracy and precision can be seen below during the training portion of our study.

Accuracy Preformance Scores			
	K- Nearest Neighbors	Random Forest	Artificial Neural Network
Standard Scalar	0.69	0.68	0.67
MinMax Scaler	0.67	0.69	0.63
Optimized Hyperparameter	0.69	0.70	0.68

Table 2. The comparison of Accuracy performance scores for each model with different ways the data was preprocessed and with the optimized hyperparameters.

Precision Preformance Scores			
	K- Nearest Neighbors	Random Forest	Artificial Neural Network
Standard Scalar	0.69	0.68	0.64
MinMax Scaler	0.68	0.70	0.61
Optimized Hyperparameter	0.68	0.70	0.68

Table 3. The comparison of Precision performance scores for each model with different ways the data was preprocessed and with the optimized hyperparameters.

3.1 K-Nearest Neighbors

When looking at the two training models for KNN, one with the MinMaxScaler and the other with the StandardScaler, we noticed that the StandardScaler produces a higher accuracy and precision score of 69% for each. Thus, we used the data that was manipulated by the StandardScaler when tuning our hyperparameters for KNN. Also, we notice that we achieve a base level of 69% accuracy before hyperparameter turning.

For KNN, there is only one parameter that needs to be tuned. This is the integer K which represents the number of neighbors the algorithm will consider when classifying a new data point. After tuning hyperparameters, we found that the most optimal K was 42 neighbors. This gave a 2% bump in accuracy and a 1% decrease in precision. Thus, the total accuracy of the final model is 70% and precision is 68%. We believe that this indeed did make the model more accurate in classifying new wines.

3.2 Random Forest

Similar to K-Nearest Neighbors above, we trained two models with each being trained using the two different types of normalized/standardized data. The first model, using the StandardScaler, produced an accuracy of 68% and a precision of 68%. The second model produced an accuracy of 69% and a precision of 70%. Thus we have an accuracy of 70% before optimizing our hyperparameters. When optimizing our hyperparameters we use the data that has been manipulated using the MinMaxScaler.

When tuning hyperparameters for Random Forest, we only need to tune two different parameters. The first one is the maximum depth a single decision tree can have within the forest and the second is the number of trees within the forest. After using GridSearch to tune these parameters, we found that a maximum depth of 20 and a total number of 700 trees should be included in the forest to achieve the most accurate and precise model. Using the optimal parameters produced an accuracy bump of 1% and the precision rate stayed the same. Thus, the Random Forest model using the optimized hyperparameters produced an accuracy and a precision of 70%.

3.3 Artificial Neural Networks

Finally, we trained our artificial neural network with one hidden layer of 700 nodes. When looking at the model that used the MinMaxScaler data, it produced an accuracy of 63% and a precision of 61%. When looking at the model that used the StandardScaler data, it produced an accuracy of 67% and a precision of 64%. Thus when tuning the hyperparameters of the model we used the data that had been manipulated using the StandardScaler method. The model has a base accuracy of 67% before hyperparameter tuning.

Artificial Neural Networks have many parameters that are changed and manipulated throughout the training and testing process. During hyperparameter optimization we optimized the activation function and, most importantly, the number of nodes within the hidden layer. After using grid search to optimize these parameters, we found that the optimal activation function was the Relu function with a total of 300 nodes in the hidden layer. This produced an accuracy bump of 1% and a precision bump of 4%. Thus the overall accuracy and precision of the model was 68%.

4 Conclusion:

In this project, three classification algorithms, KNN, ANN, and RF, were implemented using a data set on the quality of red wine. We started by utilizing basic data preprocessing techniques to manipulate the data to a usable and understandable form. After this we trained our models with two forms of scaled data. Next we found the most optimal hyperparameters by using grid search. To understand the performance of each we compared the accuracy and precision of each model. In our project we found that Random Forest had the highest accuracy of 70% and high precision at 70%. We believe that the other models, while they produced similar results, did not achieve the best level of classification based on these performance statistics. One factor that might have influenced the accuracy of our model was the fact that we merged a few classed during the data preprocessing portion of this experiment. By doing this, we are turning our model to only recognize or classify four types of wines (i.e. wines belonging to classes 5-8). Because of this, when we expose our model to a larger set of unseen data our model might not generalize well due to this class merging. If we were to continue working with this data set, we would like to have seen a more balanced data set with there being an equal number of instances within each class. By doing this, the model would be able to learn a wider variety of wines and thus generalize slightly better.

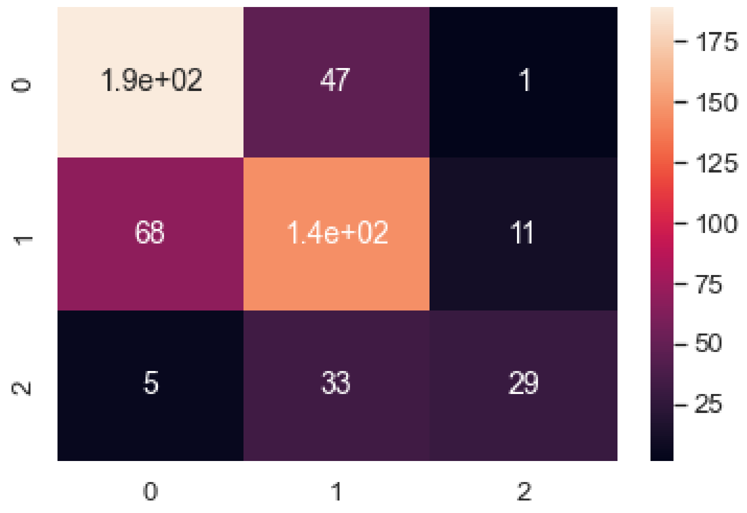
In conclusion, we believe that we successfully implemented several supervised classification models, manipulated data using typical processing techniques, and tuned hpyerparmeters to boost the performance of each model. By doing all of these, we achieve a more in depth understand about the life cycle of building and maintaining a Machine Learning model.

5 Appendix :

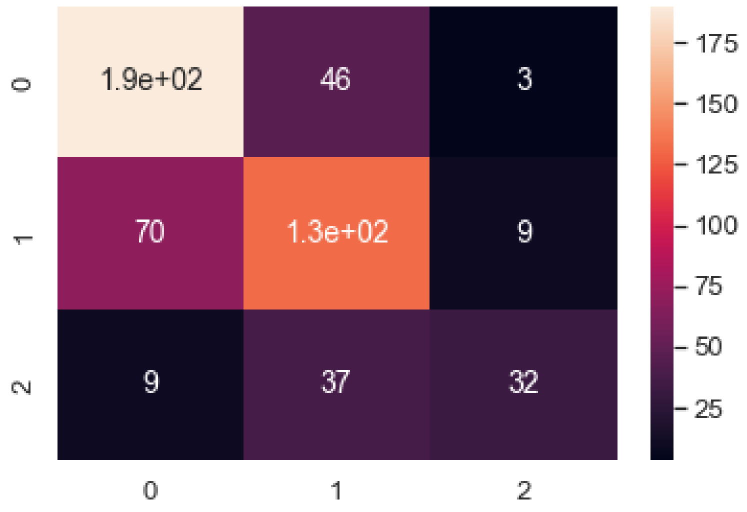
Buitinck, Lars, et al. “API Design for Machine Learning Software: Experiences from the Scikit-Learn Project.” ArXiv.org, 1 Sept. 2013, https://arxiv.org/abs/1309.0238.

Pedregosa, Fabian, et al. “Scikit-Learn: Machine Learning in Python.” Journal of Machine Learning Research, 1 Jan. 1970, https://jmlr.csail.mit.edu/papers/v12/pedregosa11a.html.

Figure 5: K- Nearest Neighbors

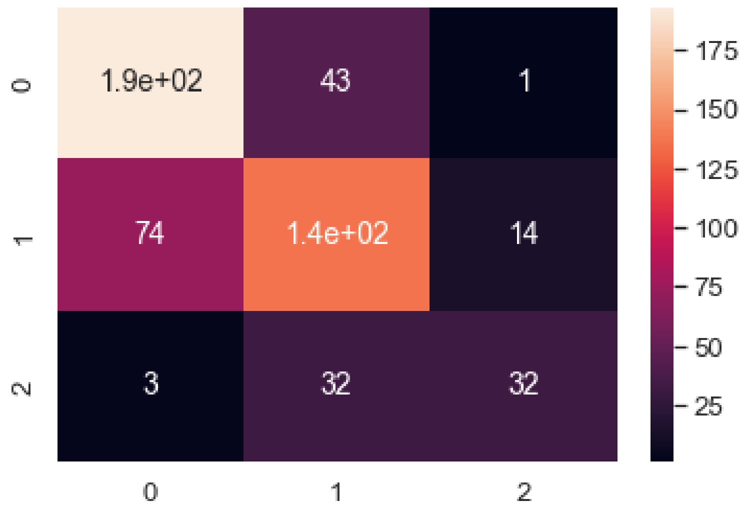


```
              precision    recall  f1-score   support

           5       0.72      0.80      0.76       237
           6       0.64      0.65      0.65       224
           7       0.71      0.43      0.54        67

    accuracy                           0.69       528
   macro avg       0.69      0.63      0.65       528
weighted avg       0.69      0.69      0.68       528
```

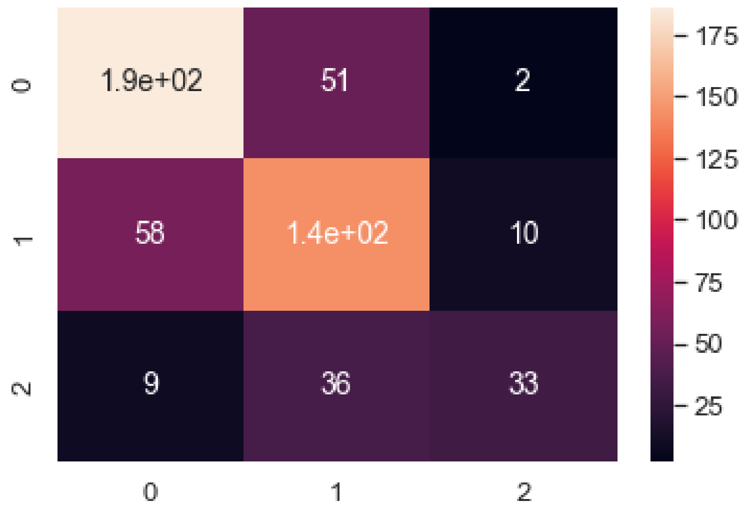


```
              precision    recall  f1-score   support

           5       0.71      0.79      0.75       239
           6       0.61      0.63      0.62       211
           7       0.73      0.41      0.52        78

    accuracy                           0.67       528
   macro avg       0.68      0.61      0.63       528
weighted avg       0.67      0.67      0.66       528
```

Figure 6: Random Forest

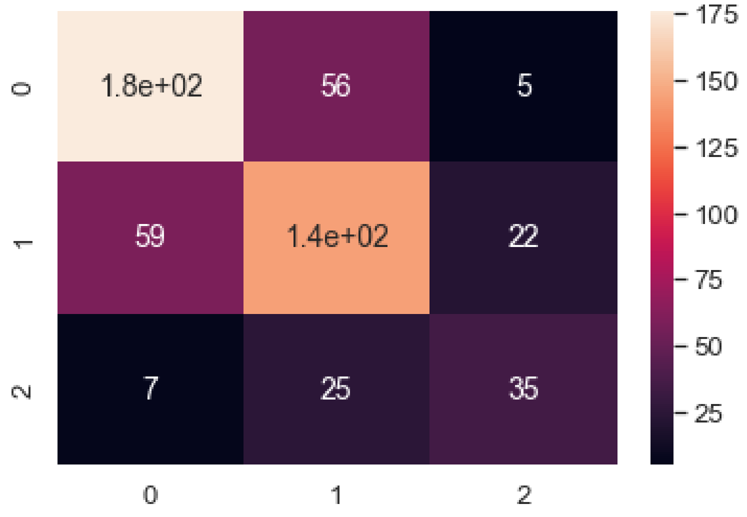


```
              precision    recall  f1-score   support

           5       0.71      0.81      0.76       237
           6       0.64      0.61      0.63       224
           7       0.68      0.48      0.56        67

    accuracy                           0.68       528
   macro avg       0.68      0.63      0.65       528
weighted avg       0.68      0.68      0.68       528
```

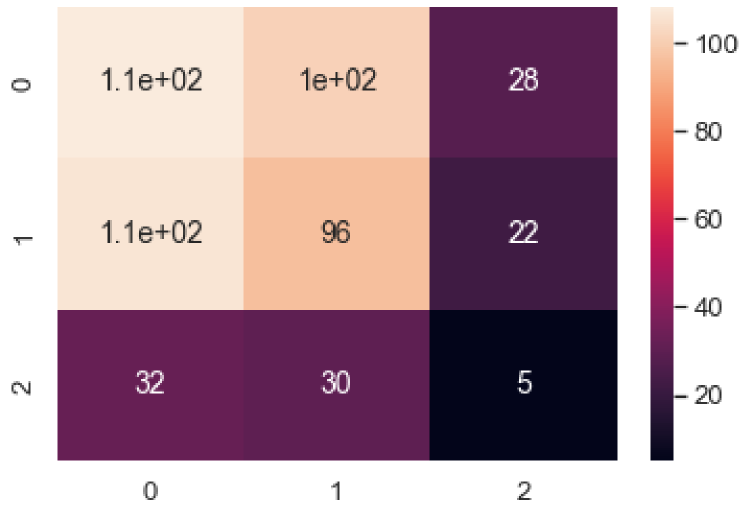


```
              precision    recall  f1-score   support

           5       0.74      0.78      0.76       239
           6       0.62      0.68      0.65       211
           7       0.73      0.42      0.54        78

    accuracy                           0.69       528
   macro avg       0.70      0.63      0.65       528
weighted avg       0.69      0.69      0.68       528
```

Figure 7: Artificial Neural Network



```
              precision    recall  f1-score   support

           5       0.73      0.74      0.73       237
           6       0.64      0.64      0.64       224
           7       0.56      0.52      0.54        67

    accuracy                           0.67       528
   macro avg       0.64      0.63      0.64       528
weighted avg       0.67      0.67      0.67       528
```



```
              precision    recall  f1-score   support

           5       0.71      0.73      0.72       239
           6       0.57      0.62      0.59       211
           7       0.56      0.40      0.47        78

    accuracy                           0.63       528
   macro avg       0.61      0.58      0.59       528
weighted avg       0.63      0.63      0.63       528
```

Final project

December 19, 2021

1 ESE417 - Introduction to Machine Learning

1.1 Using Artificial Networks and K Nearest Neighbors to Classify Red Wine Quality

1.1.1 Tillman James - Matt Donaldson - Will Li

1.1.2 December 15th, 2021

1.1.3 Data Set Description

Inputs: 1. Fixed Acidity 2. Volatile Acidity 3. Citric Acid 4. Residual Sugar 5. Chlorides 6. Free Sulfur Dioxide 7. Total Sulfur Dioxide 8. Density 9. pH 10. Sulphates 11. Alcohol

Output: Quality (Hot coding) Between 3-8

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
```

1.1.4 Importing Data

[2]:
```
redWine = pd.read_csv('winequality-red2.csv',sep = ';')
redWine.head()
```

[2]:
```
   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
```

[3]:
```
redWine.describe()
```

[3]:
```
       fixed acidity  volatile acidity  citric acid  residual sugar  \
count    1599.000000       1599.000000  1599.000000     1599.000000
mean        8.319637          0.527821     0.270976        2.538806
std         1.741096          0.179060     0.194801        1.409928
min         4.600000          0.120000     0.000000        0.900000
25%         7.100000          0.390000     0.090000        1.900000
50%         7.900000          0.520000     0.260000        2.200000
75%         9.200000          0.640000     0.420000        2.600000
max        15.900000          1.580000     1.000000       15.500000

         chlorides  free sulfur dioxide  total sulfur dioxide      density  \
count  1599.000000          1599.000000           1599.000000  1599.000000
mean      0.087467            15.874922             46.467792     0.996747
std       0.047065            10.460157             32.895324     0.001887
min       0.012000             1.000000              6.000000     0.990070
25%       0.070000             7.000000             22.000000     0.995600
50%       0.079000            14.000000             38.000000     0.996750
75%       0.090000            21.000000             62.000000     0.997835
max       0.611000            72.000000            289.000000     1.003690

                pH    sulphates      alcohol      quality
count  1599.000000  1599.000000  1599.000000  1599.000000
mean      3.311113     0.658149    10.422983     5.636023
std       0.154386     0.169507     1.065668     0.807569
min       2.740000     0.330000     8.400000     3.000000
25%       3.210000     0.550000     9.500000     5.000000
50%       3.310000     0.620000    10.200000     6.000000
75%       3.400000     0.730000    11.100000     6.000000
```

```
 max       4.010000     2.000000    14.900000     8.000000
```

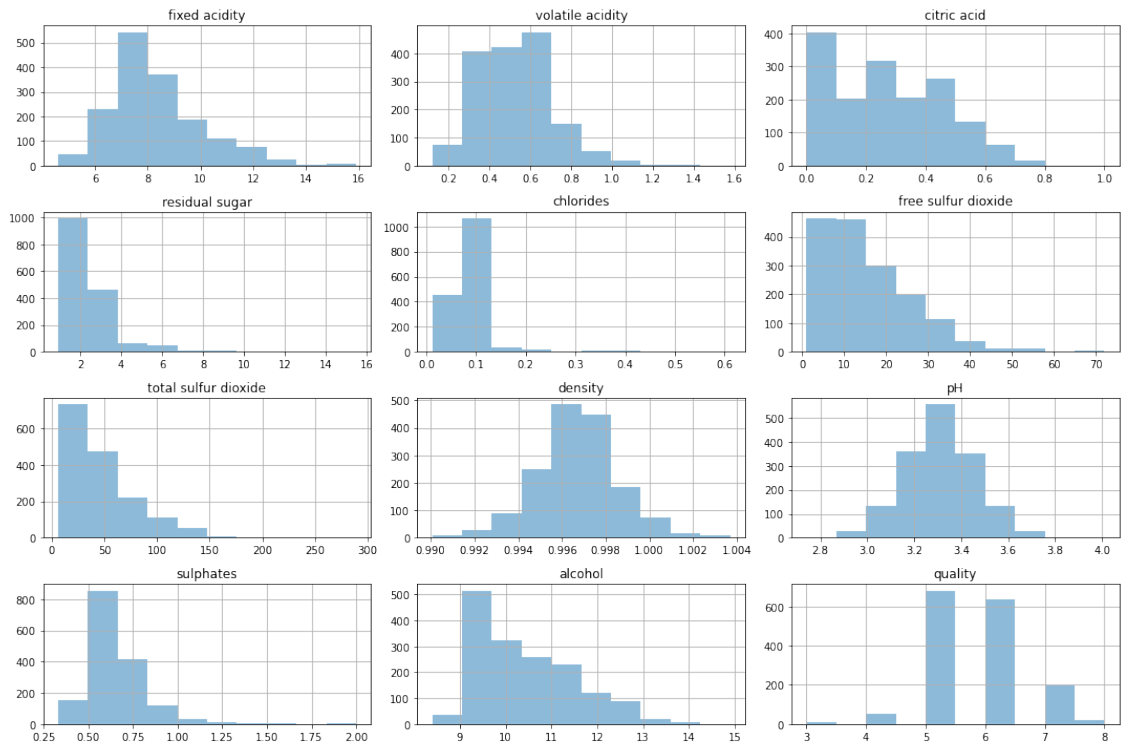
[4]:
```
redWine.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

1.1.5 PreProcessing/ Data Cleaning

Histogram for Features

[5]:
```
redWine.hist(alpha=0.5, figsize=(15, 10))
plt.tight_layout()
plt.show()
```



Dealing with the Imbalanced Dataset Looking at the quality histogram, we can see that classes 3, 4 and 8 could be noise in the dataset. In the data cleaning process, we can delete those noise in order to get a better model.

(Not necessarily noise here, we just have an inbalanced data set and need to do with this. One way to deal with this is to do what we have done and merge a few classes because "the smaller classes might just be a combination of other clases". Other ways to deal with this would be to alter our model so that the cost funciton accounts for this imbalance or we could over or under sample our data to form a secondary data set so that the smaller classes appear to have more of a precense in the new synthetic data set.)

```
[6]: redWine['quality'].value_counts()
```

```
[6]: 5    681
     6    638
     7    199
     4     53
     8     18
     3     10
     Name: quality, dtype: int64
```

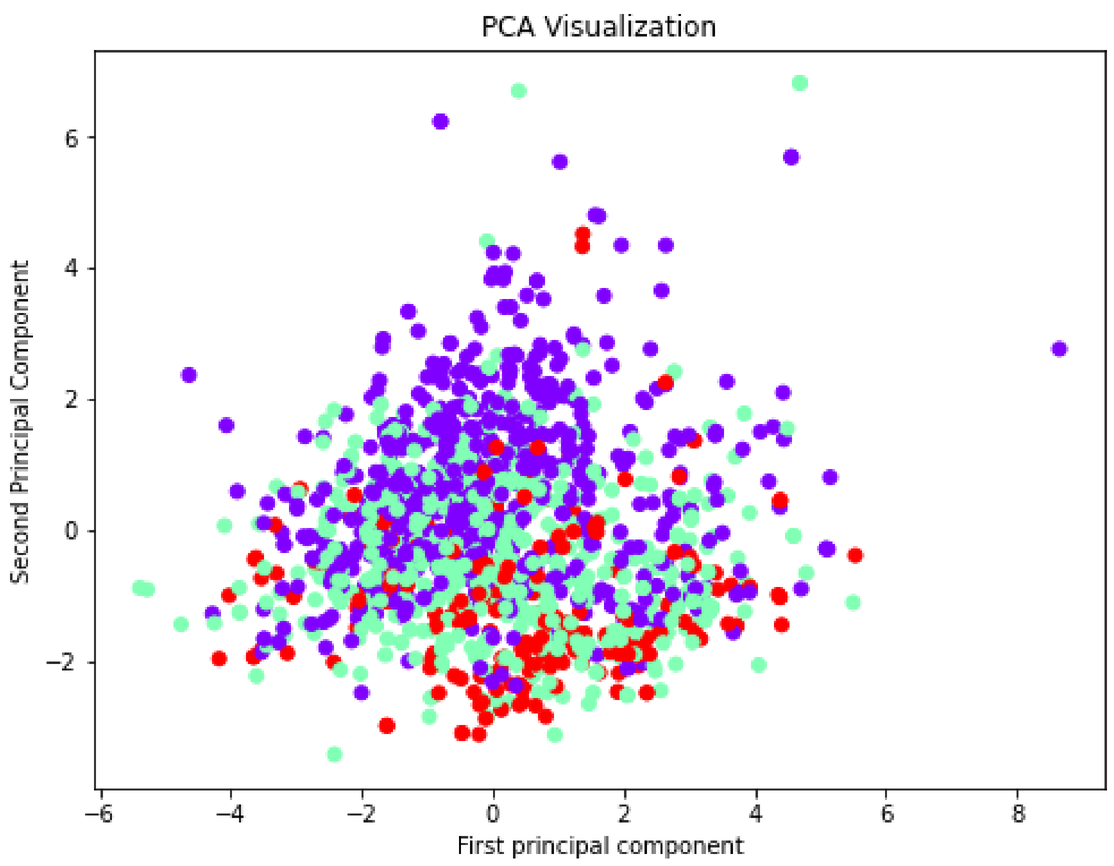
We can see here the amount of data that fall into each class. class 5,6 and 7 have the largest amount of data and classes, 4,8 and 3 have a small amount of data. So we can merge the smaller groups in order to (deal with this imbalance) make a better model.

```
[7]: for i in range(len(redWine)):
         if redWine.iloc[i, -1] == 3:
             redWine.iloc[i, -1] = 5

         if redWine.iloc[i, -1] == 4:
             redWine.iloc[i, -1] = 5

         if redWine.iloc[i, -1] == 8:
             redWine.iloc[i, -1] = 7

     redWine['quality'].value_counts()
```

```
[7]: 5    744
     6    638
     7    217
     Name: quality, dtype: int64
```

Principle Component Analysis

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

     #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("PCA Visualization")

     plt.show()
```



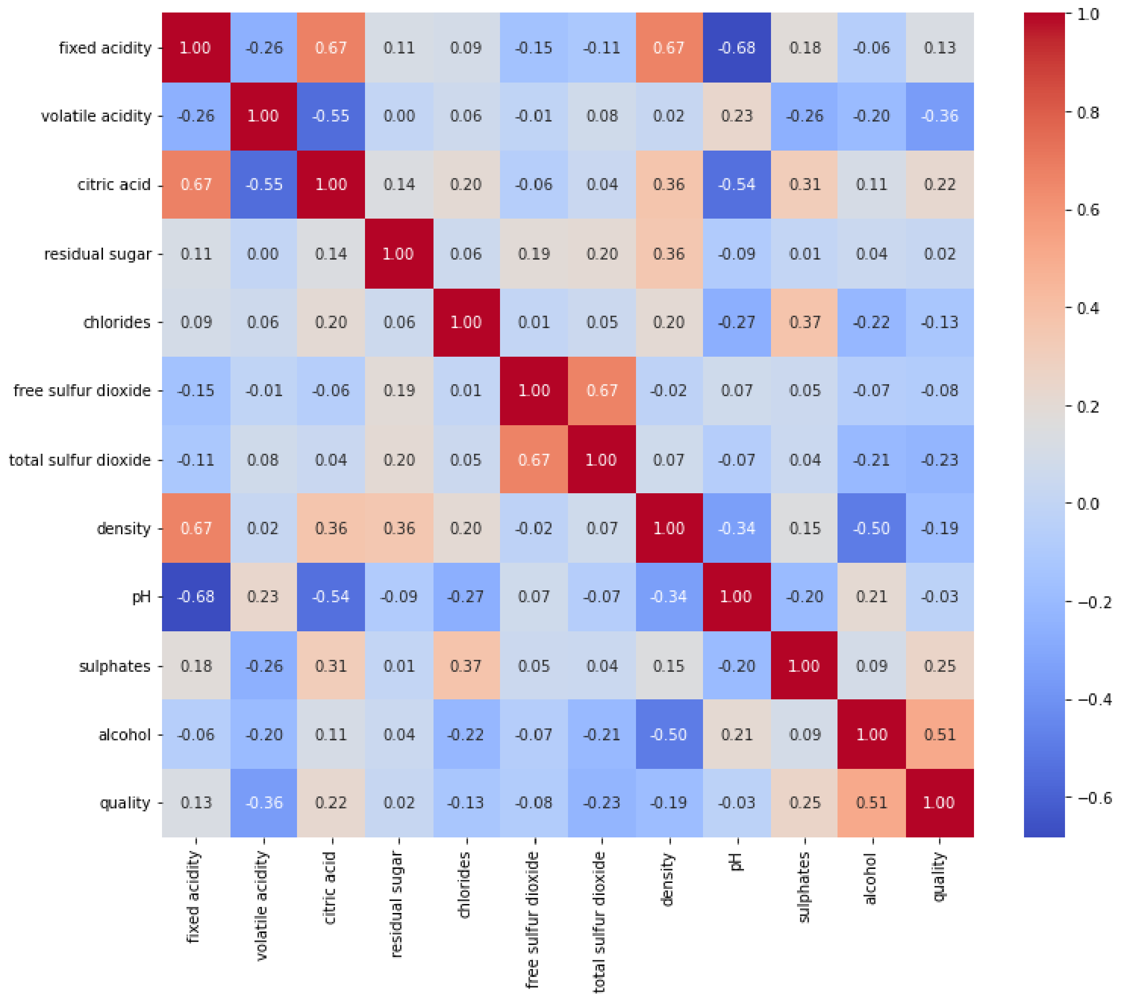
This is a very nonlinear data set no matter how you look at it in 2D.

[9]:
```
#pca.components_
print("explained variance: ", pca.explained_variance_, "\n")
exp_var_rat = pca.explained_variance_ratio_
print("explained variance ratio: ", exp_var_rat)
```

```
explained variance:  [3.10107182 1.92711489]

explained variance ratio:  [0.28173931 0.1750827 ]
```

The first principal component is only 28.2% of the variance, and the second principal component is 17.5%. So two largest principal components account for 45.7% of the variance.

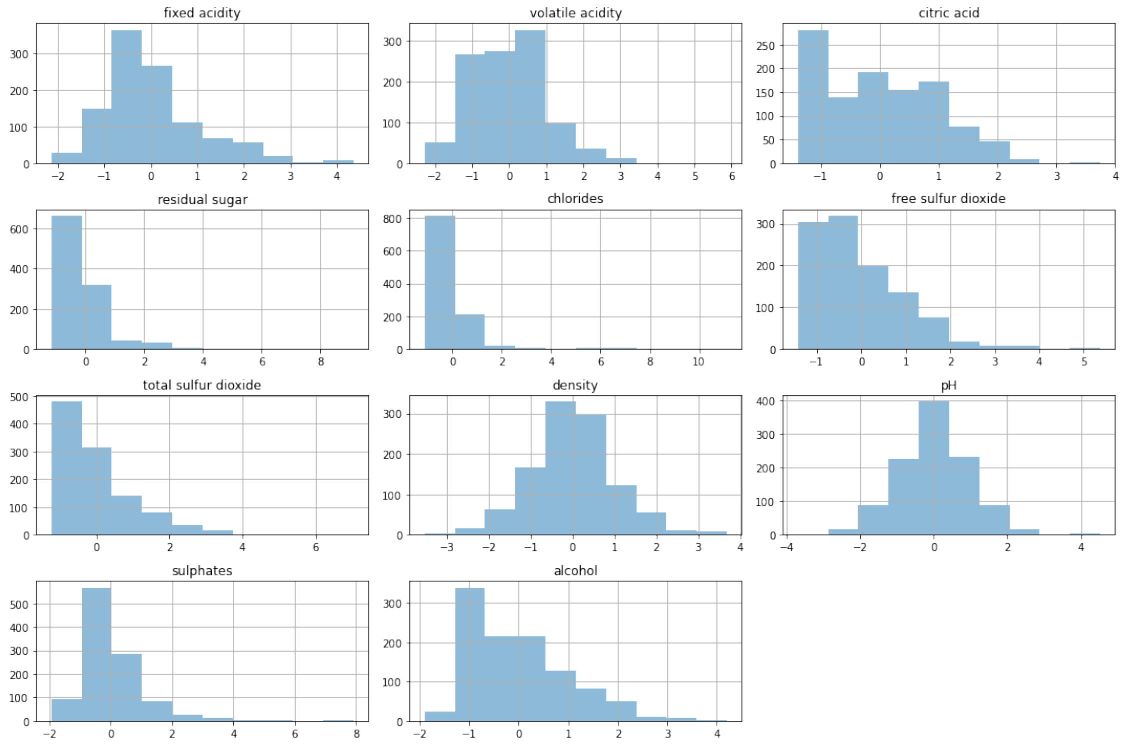
[10]:
```
plt.figure(figsize=(12,10))
sns.heatmap(redWine.corr(),annot=True, cmap='coolwarm',fmt='.2f')
plt.show()
```

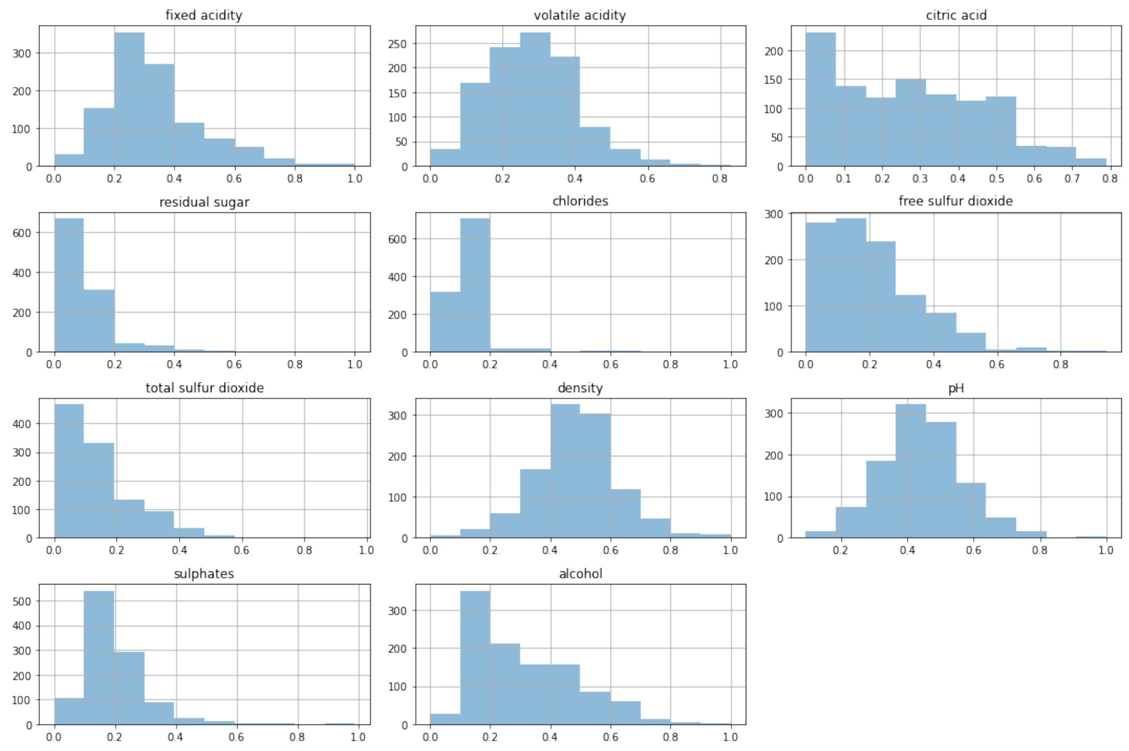


	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
fixed acidity	1.00	-0.26	0.67	0.11	0.09	-0.15	-0.11	0.67	-0.68	0.18	-0.06	0.13
volatile acidity	-0.26	1.00	-0.55	0.00	0.06	-0.01	0.08	0.02	0.23	-0.26	-0.20	-0.36
citric acid	0.67	-0.55	1.00	0.14	0.20	-0.06	0.04	0.36	-0.54	0.31	0.11	0.22
residual sugar	0.11	0.00	0.14	1.00	0.06	0.19	0.20	0.36	-0.09	0.01	0.04	0.02
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	-0.08
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	-0.23
density	0.67	0.02	0.36	0.36	0.20	-0.02	0.07	1.00	-0.34	0.15	-0.50	-0.19
pH	-0.68	0.23	-0.54	-0.09	-0.27	0.07	-0.07	-0.34	1.00	-0.20	0.21	-0.03
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	0.51
quality	0.13	-0.36	0.22	0.02	-0.13	-0.08	-0.23	-0.19	-0.03	0.25	0.51	1.00

Scaling and Normalizing Data Replotted the different features after normalization and scaling to see if it changed anything. Notice that the data is still skewed.

[11]:
```
X = redWine.drop(['quality'], axis = 1)
X_std = StandardScaler().fit_transform(X)
X_min_max = MinMaxScaler().fit_transform(X)
y = redWine['quality']

#splitting into test and training data
X_train_std, X_test_std, y_train_std, y_test_std = train_test_split(X_std, y,␣
 ↪test_size=0.33)
X_train_mm, X_test_mm, y_train_mm, y_test_mm = train_test_split(X_min_max, y,␣
 ↪test_size=0.33)

# getting column titles for label
tit = redWine.columns.values.tolist()
tit = tit[:-1]

#plotting histogram for newly scaled data
X_traind_std = pd.DataFrame(X_train_std, columns = tit)
X_traind_std.hist(alpha=0.5, figsize=(15, 10))
plt.tight_layout()
plt.title('Scaled Data')
plt.show()

#plotting histogram for newly normalized data
X_traind_mm = pd.DataFrame(X_train_mm, columns = tit)
X_traind_mm.hist(alpha=0.5, figsize=(15, 10))
plt.tight_layout()
plt.title('Normalized Data')
plt.show()
```

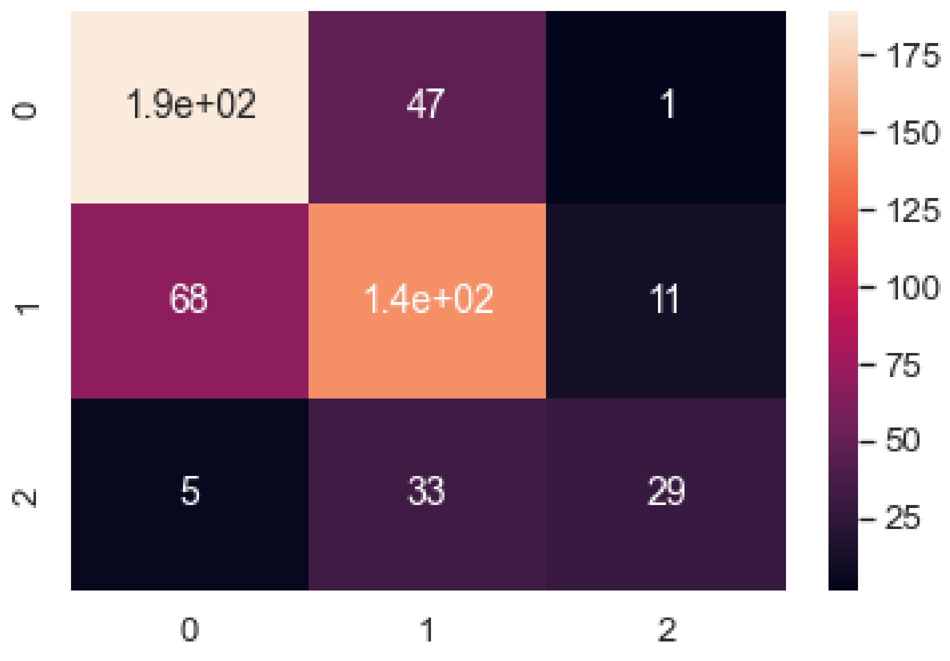




1.1.6 Classifier Methods

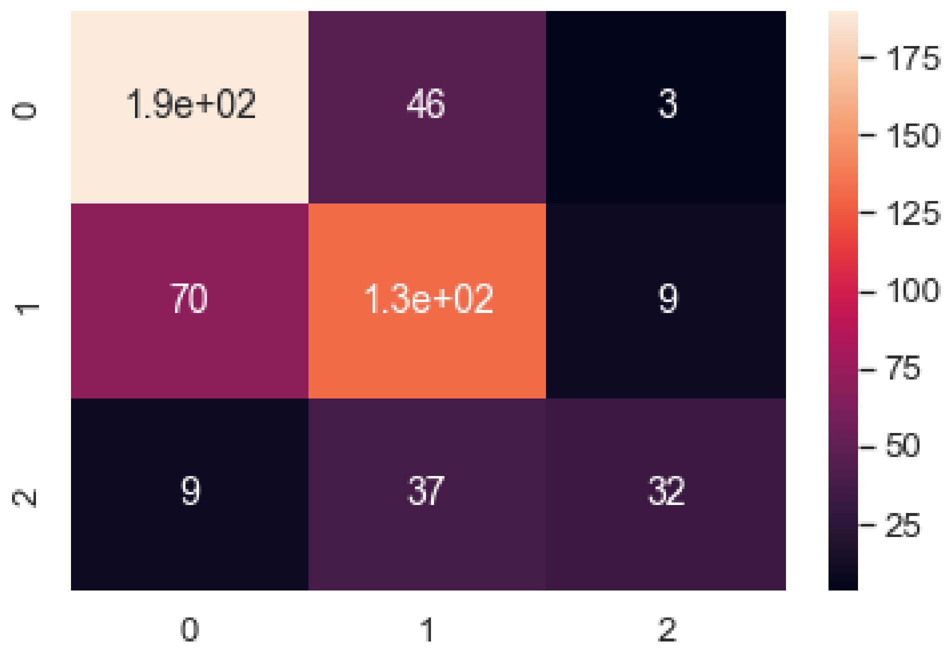
KNN

[12]:
```
# StandardScaler

classifier_knn = KNeighborsClassifier(n_neighbors = 30, weights = 'distance')
classifier_knn.fit(X_train_std,y_train_std)
pred_knn = classifier_knn.predict(X_test_std)

confusion_matrix_knn = confusion_matrix(y_test_std,pred_knn)
redWineData_cm_knn = pd.DataFrame(confusion_matrix_knn)
sns.set(font_scale=1.2) # for label size
sns.heatmap(redWineData_cm_knn, annot=True, annot_kws={"size": 14}) # font size
plt.show()

print(classification_report(y_test_std, pred_knn))
```



```
              precision    recall  f1-score   support

           5       0.72      0.80      0.76       237
           6       0.64      0.65      0.65       224
           7       0.71      0.43      0.54        67

    accuracy                           0.69       528
   macro avg       0.69      0.63      0.65       528
weighted avg       0.69      0.69      0.68       528
```

[13]:
```
# MinMaxScaler

classifier_knn = KNeighborsClassifier(n_neighbors = 30, weights = 'distance')
classifier_knn.fit(X_train_mm,y_train_mm)
pred_knn = classifier_knn.predict(X_test_mm)

confusion_matrix_knn = confusion_matrix(y_test_mm,pred_knn)
redWineData_cm_knn = pd.DataFrame(confusion_matrix_knn)
sns.set(font_scale=1.2) # for label size
sns.heatmap(redWineData_cm_knn, annot=True, annot_kws={"size": 14}) # font size
plt.show()

print(classification_report(y_test_mm, pred_knn))
```

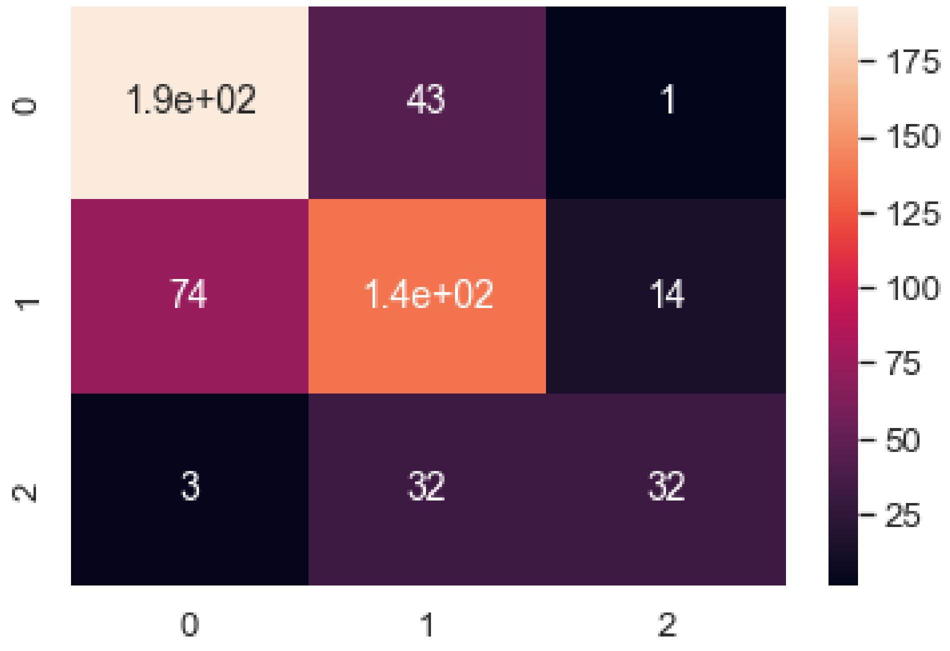


```
              precision    recall  f1-score   support

           5       0.71      0.79      0.75       239
           6       0.61      0.63      0.62       211
           7       0.73      0.41      0.52        78

    accuracy                           0.67       528
   macro avg       0.68      0.61      0.63       528
weighted avg       0.67      0.67      0.66       528
```

Random Forest

[14]:
```
# StandardScaler

classifier_rf = RandomForestClassifier(n_estimators=200, max_depth=100, ␣
 ↪max_features='sqrt')
classifier_rf.fit(X_train_std, y_train_std)
pred_rf = classifier_rf.predict(X_test_std)


confusion_matrix_rf = confusion_matrix(y_test_std, pred_rf)
redWineData_cm_rf = pd.DataFrame(confusion_matrix_rf)
sns.set(font_scale=1.2) # for label size
sns.heatmap(redWineData_cm_rf, annot=True, annot_kws={"size": 14}) # font size
plt.show()
```

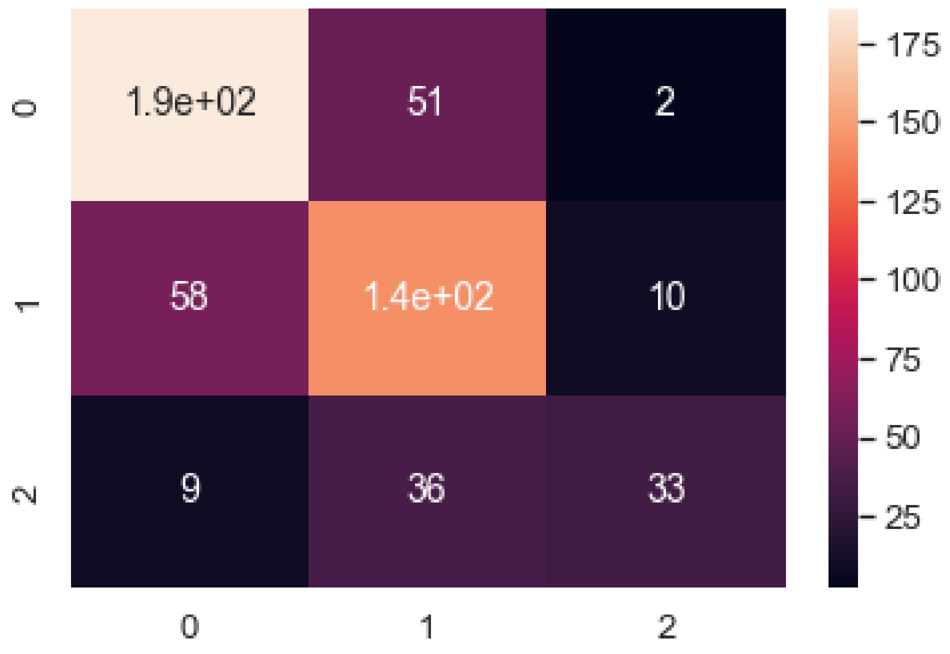
```
print(classification_report(y_test_std, pred_rf))
```



```
              precision    recall  f1-score   support

           5       0.71      0.81      0.76       237
           6       0.64      0.61      0.63       224
           7       0.68      0.48      0.56        67

    accuracy                           0.68       528
   macro avg       0.68      0.63      0.65       528
weighted avg       0.68      0.68      0.68       528
```

[15]:
```
# MinMaxScaler

from sklearn.ensemble import RandomForestClassifier
classifier_rf = RandomForestClassifier(n_estimators=200, max_depth=100, ␣
 ↪max_features='sqrt')
classifier_rf.fit(X_train_mm, y_train_mm)
pred_rf = classifier_rf.predict(X_test_mm)


confusion_matrix_rf = confusion_matrix(y_test_mm, pred_rf)
redWineData_cm_rf = pd.DataFrame(confusion_matrix_rf)
```

```python
sns.set(font_scale=1.2) # for label size
sns.heatmap(redWineData_cm_rf, annot=True, annot_kws={"size": 14}) # font size
plt.show()

print(classification_report(y_test_mm, pred_rf))
```

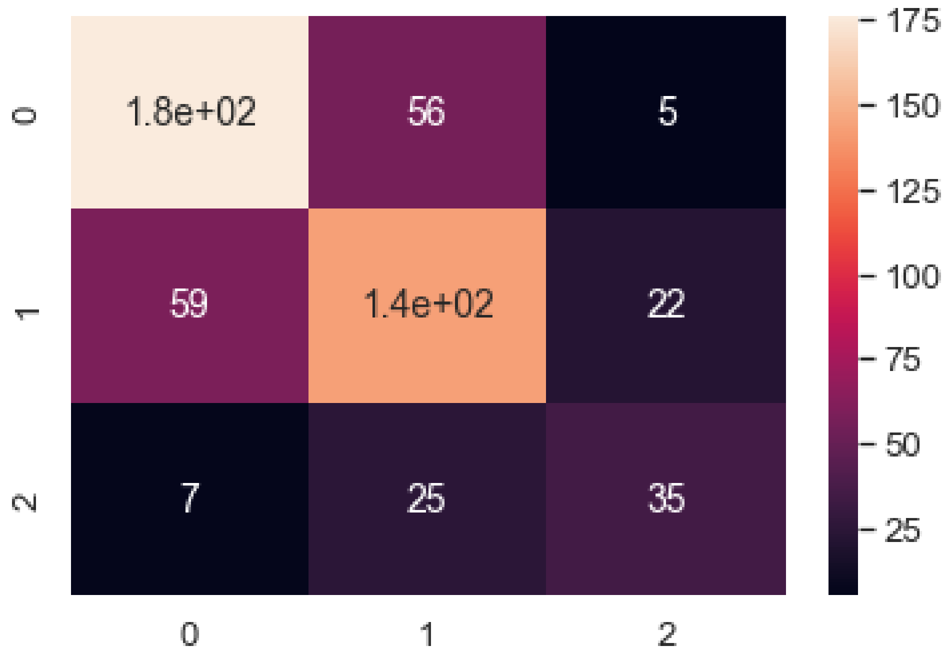


```
              precision    recall  f1-score   support

           5       0.74      0.78      0.76       239
           6       0.62      0.68      0.65       211
           7       0.73      0.42      0.54        78

    accuracy                           0.69       528
   macro avg       0.70      0.63      0.65       528
weighted avg       0.69      0.69      0.68       528
```

Artificial Neural Network

```python
[16]: # StandardScaler

classifier_ann = MLPClassifier(solver='lbfgs', activation='relu',␣
 ↪hidden_layer_sizes = 700)
classifier_ann.fit(X_train_std, y_train_std)
pred_ann = classifier_ann.predict(X_test_std)
```

```
confusion_matrix_ann = confusion_matrix(y_test_std,pred_ann)
redWineData_cm_ann = pd.DataFrame(confusion_matrix_ann)
sns.set(font_scale=1.2) # for label size
sns.heatmap(redWineData_cm_ann, annot=True, annot_kws={"size": 14}) # font size
plt.show()

print(classification_report(y_test_std, pred_ann))
```

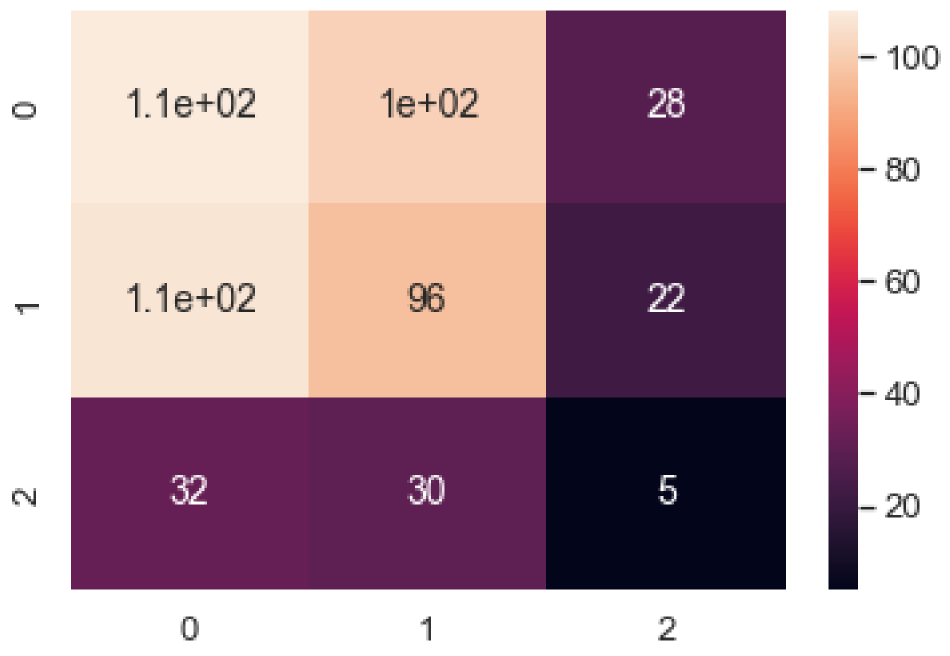


```
              precision    recall  f1-score   support

           5       0.73      0.74      0.73       237
           6       0.64      0.64      0.64       224
           7       0.56      0.52      0.54        67

    accuracy                           0.67       528
   macro avg       0.64      0.63      0.64       528
weighted avg       0.67      0.67      0.67       528
```

[17]:
```
# MinMaxScaler

from sklearn.neural_network import MLPClassifier
classifier_ann = MLPClassifier(solver='lbfgs', activation='relu',␣
 ↪hidden_layer_sizes = 700)
classifier_ann.fit(X_train_mm, y_train_mm)
```

```
pred_ann = classifier_ann.predict(X_test_mm)

confusion_matrix_ann = confusion_matrix(y_test_std,pred_ann)
redWineData_cm_ann = pd.DataFrame(confusion_matrix_ann)
sns.set(font_scale=1.2) # for label size
sns.heatmap(redWineData_cm_ann, annot=True, annot_kws={"size": 14}) # font size
plt.show()

print(classification_report(y_test_mm, pred_ann))
```

```
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:500:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  self.n_iter_ = _check_optimize_result("lbfgs", opt_res, self.max_iter)
```



```
              precision    recall  f1-score   support

           5       0.71      0.73      0.72       239
           6       0.57      0.62      0.59       211
           7       0.56      0.40      0.47        78

    accuracy                           0.63       528
```

```
   macro avg       0.61      0.58      0.59       528
weighted avg       0.63      0.63      0.63       528
```

1.1.7 Hyperparameter Tuning - Grid Search

KNN Grid Search

[24]:
```
redWine_knn = KNeighborsClassifier()
k = list(range(2,400,10))

#create a list of parameters to tune
param_grid_knn = dict(n_neighbors =  k)

#fit the model using grid search
CV_knn = GridSearchCV(estimator = redWine_knn, param_grid=param_grid_knn, cv= 6)
CV_knn.fit(X_train_std, y_train_std)

#print the result of best hyperparameters
print(CV_knn.best_params_)
#redWine_knn.get_params().keys()
```

```
{'n_neighbors': 42}
```

Random Forest Grid Search

[20]:
```
redWine_rfc = RandomForestClassifier(random_state=417)

#create a list of parameters to tune
param_grid_rfc = {
    'n_estimators': [500, 600, 700, 800],
    'max_depth': [10, 15, 20, 25, 30]
}

#fit the model using grid search
CV_rfc = GridSearchCV(estimator=redWine_rfc, param_grid=param_grid_rfc, cv= 6)
CV_rfc.fit(X_train_mm, y_train_mm)

#print the result of best hyperparameters
print(CV_rfc.best_params_)
```

```
{'max_depth': 20, 'n_estimators': 700}
```

Artificial Neural Network Grid Search

[21]:
```
redWine_ann = MLPClassifier(random_state=417)

#create a list of parameters to tune
param_grid_ann = {
    'activation': ['identity', 'logistic', 'tanh', 'relu'],
    'hidden_layer_sizes' : [50,100,150,200,300],
    'max_iter' :[300]
```

```python
}

#fit the model using grid search
CV_ann = GridSearchCV(estimator=redWine_ann, param_grid=param_grid_ann, cv= 6)
CV_ann.fit(X_train_std, y_train_std)

#print the result of best hyperparameters
print(CV_ann.best_params_)
## Returns the parameters that we can optimize over
#redWine_ann.get_params().keys()
```

```
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
```

```
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
```

```
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
```

```
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
```

```
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
```

```
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

```
{'activation': 'relu', 'hidden_layer_sizes': 300, 'max_iter': 300}
```

```
/opt/homebrew/Caskroom/miniforge/base/envs/ML417/lib/python3.9/site-
```

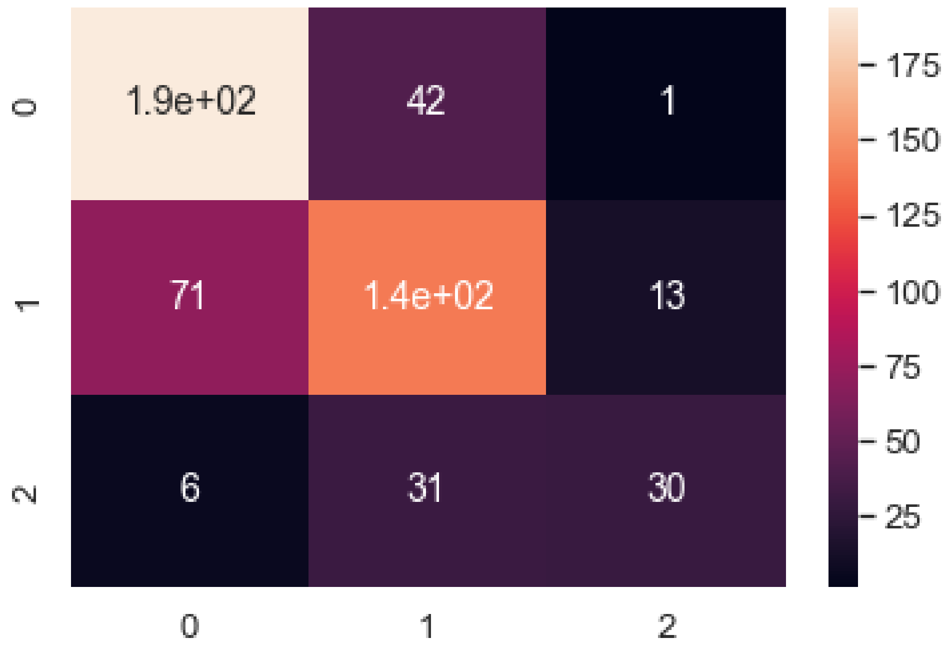
```
packages/sklearn/neural_network/_multilayer_perceptron.py:614:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (300) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

1.1.8 Model Implementation with Optimized Hyperparameters

KNN

```
[27]: print(CV_knn.best_params_.get('n_neighbors'))
classifier_knn = KNeighborsClassifier(n_neighbors = CV_knn.best_params_.
 ↪get('n_neighbors'), weights = 'distance')
classifier_knn.fit(X_train_std,y_train_std)
pred_knn = classifier_knn.predict(X_test_std)

confusion_matrix_knn = confusion_matrix(y_test_std,pred_knn)
redWineData_cm_knn = pd.DataFrame(confusion_matrix_knn)
sns.set(font_scale=1.2) # for label size
sns.heatmap(redWineData_cm_knn, annot=True, annot_kws={"size": 14}) # font size
plt.show()

print(classification_report(y_test_std, pred_knn))
```

```
42
```

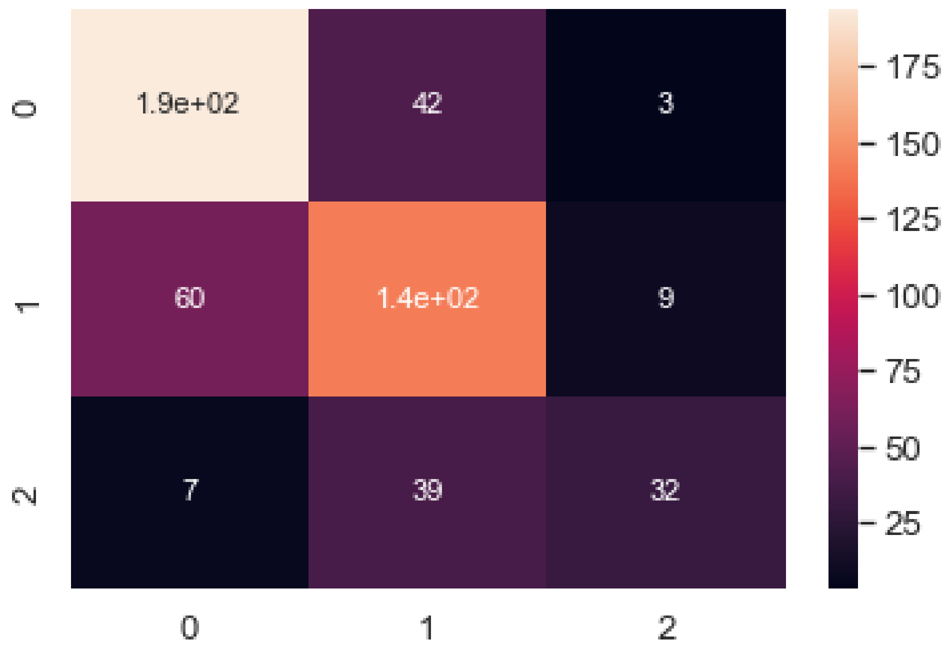


```
              precision    recall  f1-score   support

           5       0.72      0.82      0.76       237
           6       0.66      0.62      0.64       224
           7       0.68      0.45      0.54        67

    accuracy                           0.69       528
   macro avg       0.68      0.63      0.65       528
weighted avg       0.69      0.69      0.68       528
```

Random Forest

[25]:
```
print(CV_rfc.best_params_.get('n_estimators'))
print(CV_rfc.best_params_.get('max_depth'))

classifier_rf = RandomForestClassifier(n_estimators=CV_rfc.best_params_.
 ↪get('n_estimators'),
                                       max_depth=CV_rfc.best_params_.
 ↪get('max_depth'),
                                       max_features='sqrt')
classifier_rf.fit(X_train_mm, y_train_mm)
pred_rf = classifier_rf.predict(X_test_mm)


confusion_matrix_rf = confusion_matrix(y_test_mm, pred_rf)
redWineData_cm_rf = pd.DataFrame(confusion_matrix_rf)
sns.set(font_scale=1.2) # for label size
sns.heatmap(redWineData_cm_rf, annot=True, annot_kws={"size": 11}) # font size
plt.show()

print(classification_report(y_test_mm, pred_rf))
```

```
700
20
```



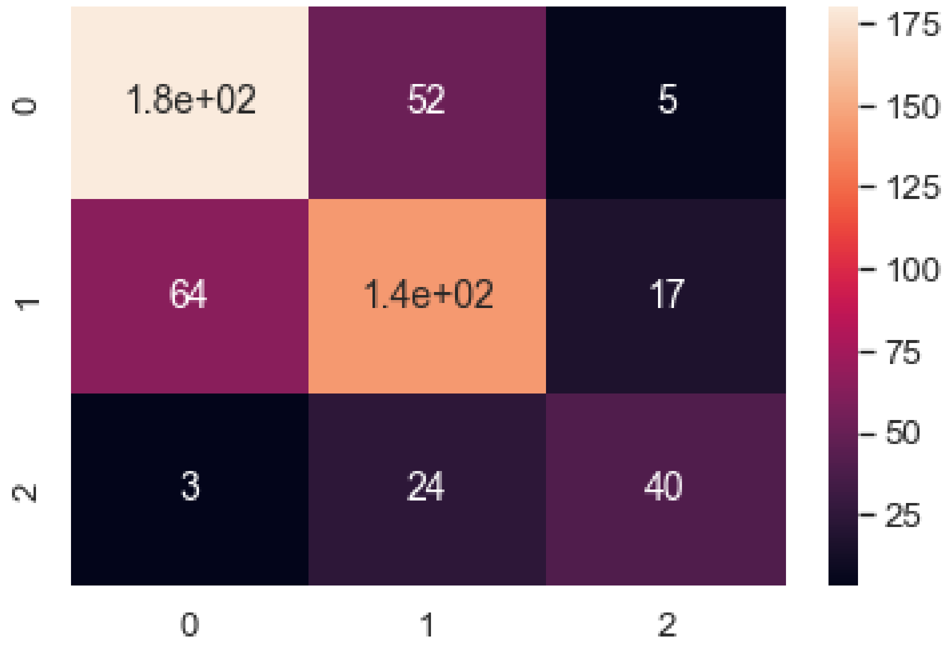
```
              precision    recall  f1-score   support

           5       0.74      0.81      0.78       239
           6       0.64      0.67      0.65       211
           7       0.73      0.41      0.52        78

    accuracy                           0.70       528
   macro avg       0.70      0.63      0.65       528
weighted avg       0.70      0.70      0.69       528
```

Artificial Neural Netowrk

[26]:
```
print(CV_ann.best_params_.get('activation'))
print(CV_ann.best_params_.get('hidden_layer_sizes'))
print(CV_ann.best_params_.get('max_iter'))

classifier_ann = MLPClassifier(solver='lbfgs',
                               activation=CV_ann.best_params_.get('activation'),
                               hidden_layer_sizes = CV_ann.best_params_.
 ↪get('hidden_layer_sizes'),
                               max_iter = CV_ann.best_params_.get('max_iter'))
classifier_ann.fit(X_train_std, y_train_std)
pred_ann = classifier_ann.predict(X_test_std)

confusion_matrix_ann = confusion_matrix(y_test_std,pred_ann)
```

```
redWineData_cm_ann = pd.DataFrame(confusion_matrix_ann)
sns.set(font_scale=1.2) # for label size
sns.heatmap(redWineData_cm_ann, annot=True, annot_kws={"size": 14}) # font size
plt.show()

print(classification_report(y_test_std, pred_ann))
```

```
relu
300
300
```



```
              precision    recall  f1-score   support

           5       0.73      0.76      0.74       237
           6       0.65      0.64      0.65       224
           7       0.65      0.60      0.62        67

    accuracy                           0.69       528
   macro avg       0.68      0.66      0.67       528
weighted avg       0.69      0.69      0.69       528
```

```
[ ]:
```