Classification of Red Wine Quality using Various Machine Learning Algorithms

1st Haoyu Quan
Mckelvey School of Engineering
Washington University in St. Louis
St. Louis, MO
quanhaoyu@wustl.edu

2nd Anny Qiao
Mckelvey School of Engineering
Washington University in St. Louis
St. Louis, MO
a.qiao@wustl.edu

3rd Bruce Li
Mckelvey School of Engineering
Washington University in St. Louis
St. Louis, MO
liyifei@wustl.edu

***Abstract*—This study implemented three machine learning classification methods, KNN, SVM, and Random Forest, on the UCI red-wine dataset to classify wines. Results showed that the KNN method had the lowest performance with an accuracy of 61.97% and an MSE of 0.4985, while the SVM method performed better with an accuracy of 65.87% and an MSE of 0.4678. The Random Forest method achieved the highest accuracy of 75% and the lowest MSE of 0.35. The study highlights the importance of data cleaning and standardization in improving the accuracy and MSE of machine learning models. The findings suggest that the Random Forest method is the most effective approach for wine classification.**

I. Introduction

Machine learning is a branch of artificial intelligence that enables machines to learn from data and make predictions or decisions without being explicitly programmed. It finds applications in a wide range of domains such as image recognition, natural language processing, and predictive modeling. In this project, we aim to use supervised machine learning algorithms to classify a wine quality data set based on 11 given features. While wine was once viewed as a luxury good, it is now enjoyed by a wider range of consumers. Quality evaluation is a crucial part of the certification process and can help to identify influential factors during the wine production process. [1]In our project, these factors will be used as features for our machine learning algorithm.

The primary objective of this project is to build a machine learning model that can accurately predict the quality of wine into 10 different levels with the given data. Furthermore, we aim to compare the performance of different machine learning algorithms such as Support Vector Machines (SVM), K-Nearest Neighbors (KNN), and Random Forest to determine the most effective algorithm for this task. To improve the accuracy of our model, we have implemented various methods of data cleaning, chosen optimal weight factors, and tuned other hyperparameters.

After multiple testing and trial and error iterations, the model was able to achieve an accuracy rate of over 60 percent. Overall, this project demonstrates the effectiveness of supervised machine learning algorithms in predicting the quality of wine based on various influential factors during the production process.

II. Methods

A. Exploratory Data Analysis

The dataset used in this project was introduced by Cortez et al. [2]

The dataset is a collection of 1599 samples, each representing a different red wine, with 11 physicochemical attributes that describe the properties of the wine accompanied with a quality rating. These attributes includes but not limited to: amount of various acid, residual sugar, chlorides, sulfates in the wine; density; pH value; alcohol content; etc. The quality rating is scored as an integer between 0 and 9.

As the original research aims to find which chemical properties influence the quality of red wines, we here focus on analyzing the column of wine quality.

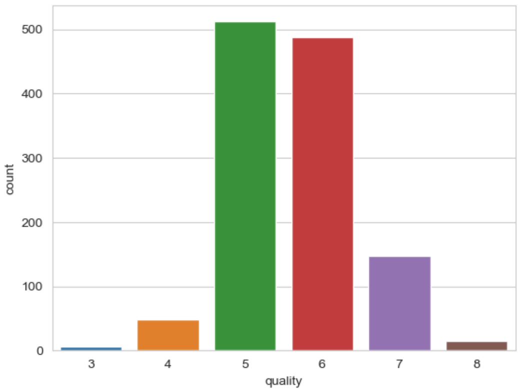


Fig. 1. Quality distribution of the dataset

As shown in figure 1, according to the samples, wine scores are in range from 3 to 8, and most of them have a score of 5 and 6.

From the heat map in figure 2, we can conclude that alcohol, volatile acidity, citric acid and sulfates have most correlated attributes with quality. Besides that, the heat map shows that alcohol has a weak positive correlation with the pH value, citric acid and density have a strong positive correlation with fixed acidity, and pH has a negative correlation with density, fixed acidity, citric acid, and sulfates.

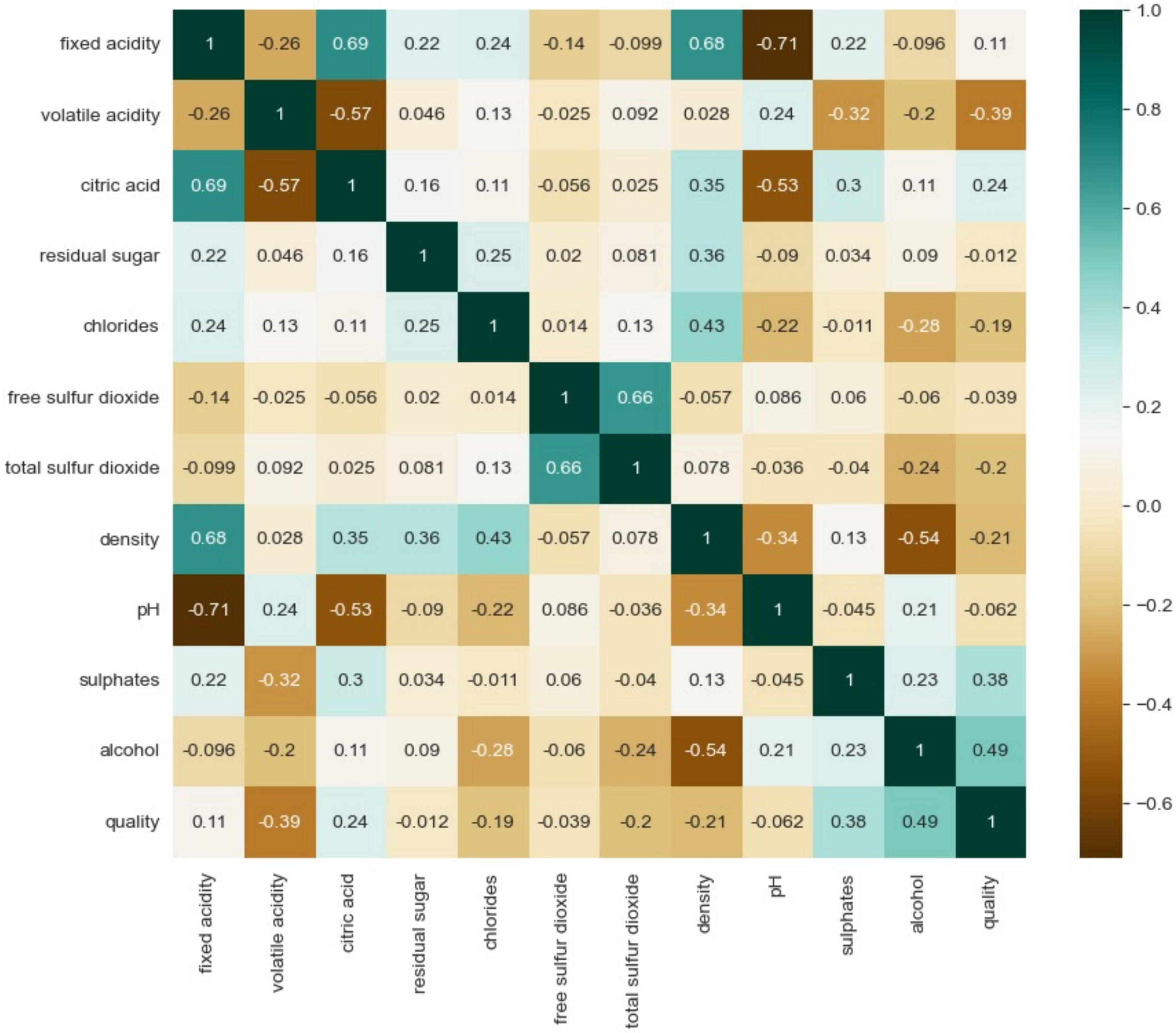


Fig. 2. Heat map of correlations between variables

The box plots of selected feature variable over different rated quality is shown as well in figure 3 to 6, which implicates same insights on correlation between physiochemical variable and quality as summarized above.

B. Data Cleaning

As the first step of the data cleaning process, duplicates are removed. Including duplicates will lead to the model overfitting this subset of points. Besides that, data leakage from validation to training data can occur due to duplicates present in both sets, which would lead to an fictitious increase in validation performance of the model since duplicates were already learned during training.

Furthermore, outliers in the dataset are removed as the presence of outliers can affect the accuracy of statistical models. Outliers might cause the model to skew towards them, resulting in an inaccurate representation of the underlying data distribution. This can lead to overfitting, where the model fits the training data too closely and performs poorly on new, unseen data. Especially, outliers affect the performance of models based on distance measures, such as k-nearest neighbors. In KNN, outliers can be misclassified or affect the clustering of other data points, leading to inaccurate results.

C. Grid Search and Weighted Features

Grid search is a widely used hyperparameter optimization technique in machine learning that involves systematically evaluating a range of hyperparameters for a given model. This technique entails creating a grid of all possible combinations of hyperparameters and training a model for each combination. The optimal combination of hyperparameters is then determined by selecting the one that yields the highest accuracy or lowest error rate, depending on the specific problem being addressed.

In the present project, grid search was employed to fine-tune the hyperparameters of the Support Vector Machine (SVM)

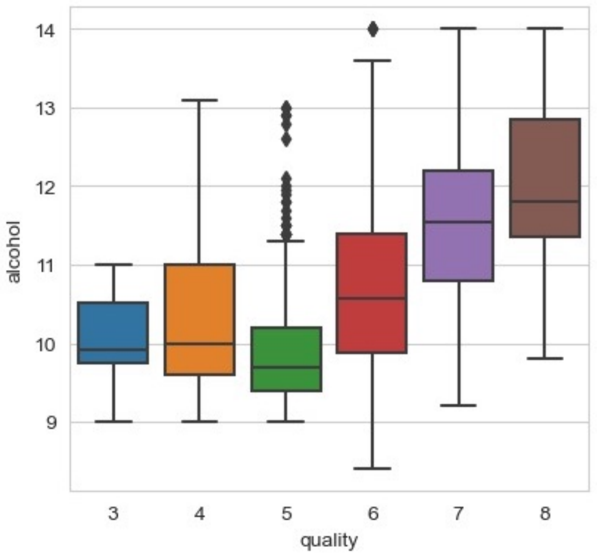


Fig. 3. Box plot of alcohol over quality

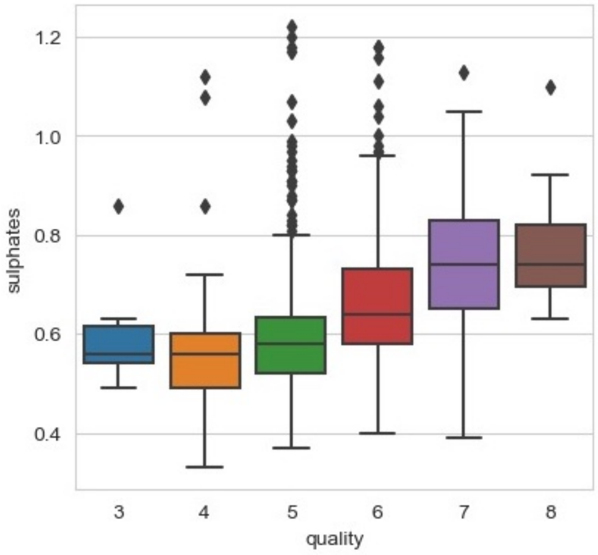


Fig. 4. Box plot of sulohates over quality

model. Specifically, the hyperparameters C and gamma were tuned using grid search to optimize the model's performance. These two hyperparameters are of particular importance in determining the efficacy of the SVM model in handling the data at hand. [5]

To accomplish this, a two-stage approach was used, where a broad range of C and gamma values was initially explored, followed by a more focused search of a smaller region to identify the optimal values for these hyperparameters. The implementation of the grid search algorithm used in this project is presented below:

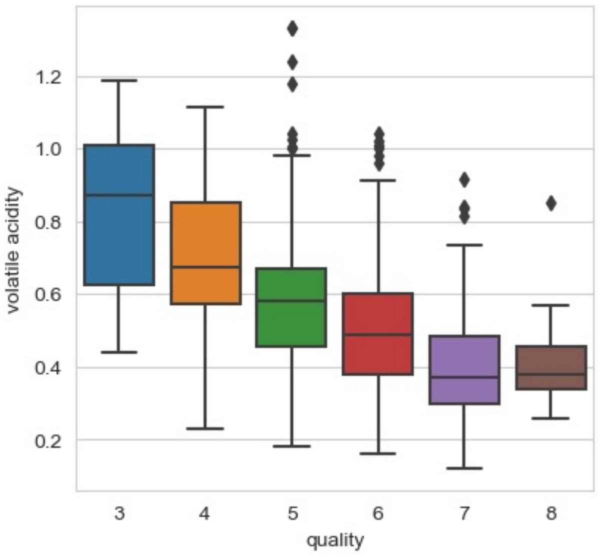


Fig. 5. Box plot of volatile acidity over quality

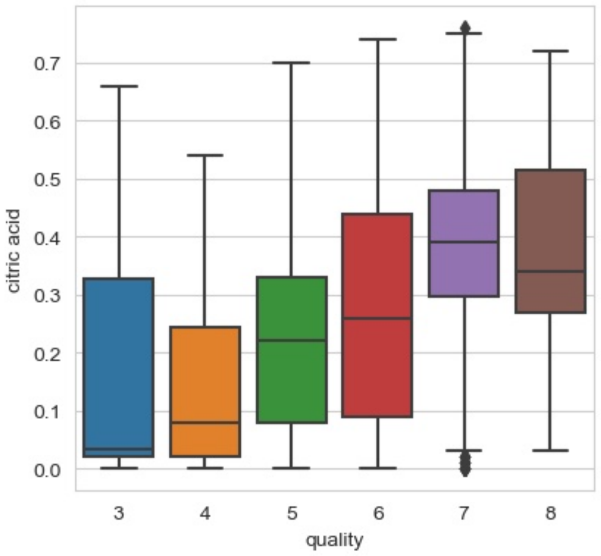
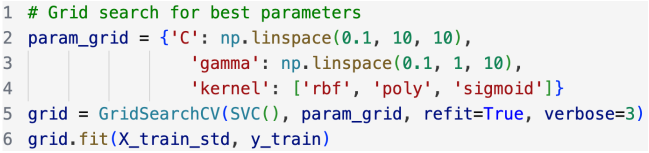


Fig. 6. Box plot of citric acid over quality

D. KNN

K-Nearest Neighbors (KNN) is a supervised learning algorithm that can be utilized for both classification and regression tasks. The fundamental principle of KNN is to classify a data point based on the class labels of its K-nearest neighbors in the feature space. The choice of K, as well as the metric used for calculating distance, are important factors that influence the performance of the model. [6]



```
# Grid search for best parameters
param_grid = {'C': np.linspace(0.1, 10, 10),
              'gamma': np.linspace(0.1, 1, 10),
              'kernel': ['rbf', 'poly', 'sigmoid']}
grid = GridSearchCV(SVC(), param_grid, refit=True, verbose=3)
grid.fit(X_train_std, y_train)
```

Fig. 7. Grid Search Code

E. SVM

The Support Vector Machine (SVM) is a supervised machine learning algorithm that has gained popularity for its success in various classification tasks, including wine quality classification. SVM works by identifying a decision boundary that effectively separates the different categories of wine quality in the feature space.

To classify the wine quality, the SVM requires input features such as alcohol content, acidity, sugar content, and other sensory attributes. The objective of the SVM is to find a hyperplane or a linear decision boundary that can best separate the various quality categories of wine samples in the feature space. The dimensionality of this hyperplane is higher than the dimensionality of the data, and it is determined by selecting a marginal maximization hyperplane. This approach ensures that the SVM classifier is robust by calculating the distance between the hyperplane and the nearest data point for each quality category.

However, in cases where the data is not linearly separable, SVMs utilize kernel functions to transform the data to a higher dimensional space, which makes it linearly separable. Among the commonly used kernel functions for wine quality classification are linear, polynomial, radial basis function (RBF), and sigmoid kernels. These kernels map the data into a new feature space, where the SVM can find a hyperplane that separates the different quality categories with a good margin. [3]

In this project, SVM has been trained with raw data then with the standardized and cleaned data. Then Grid Search is implemented to find the best hyperparameters of C and gamma. With all the methods implemented with SVM model, the accuracy has a 10% improvement and MSE has a 33% decrement.

F. Random Forest

Random Forest is an ensemble learning algorithm that is used for classification tasks. It works by constructing multiple decision trees and then combining their predictions to make a final prediction. The algorithm randomly selects a subset of features and a subset of training samples for each decision tree, which helps to reduce overfitting and improve the performance of the model.Also, to simplify the process, the sklearn method of $RandomForestClassifier()$ is used to classify the wine quality. (sklearn). There are eleven features imputed to the ML algorithm. Because the correlation of the wine quality is more related to some features than others, the features need to be considered differently while imputed into the machine learning algorithm. Here are the weights for each feature used in the program :1: 10, 2: 10, 3: 69, 4: 1, 5: 1, 6: 1, 7: 1, 8: 34, 9: 10, 10: 10, 11: 80. Those weights are estimated from the input importance of the people originally conducting the experiments. [4]

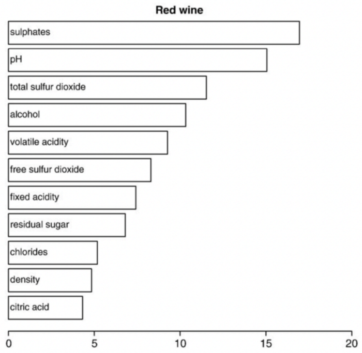


Fig. 8. Red Wine Weight

Due to their great variety of hyperparameters, to optimize the accuracy of the method four are used which are $nestimators, maxdepth, randomstate, classweight.$ The grid search is implemented to come up with the best combination of the hyperparameters, $'maxdepth' : 15, 'nestimators' : 700$. With given depth and number of estimators the Machine learning algorithm would output the best performances.

III. Result and Analysis

Implementing the three method mentioned above shows Random Forest does the best in all the methods.

A. KNN

KNN method was implemented with different K number, and the accuracy result is shown below: In the context of K-Nearest Neighbor (KNN) algorithm, the relationship between the value of K and the accuracy of both the training and test sets was investigated. The results showed that as K value increases, the accuracy of the training set decreases while the accuracy of the test set increases. A low value of K can cause overfitting of the data to the test set, leading to poor generalization. Conversely, increasing the K value leads to the convergence of the test accuracy and training accuracy, indicating that the KNN model is well trained. These findings suggest that the choice of K value is an important consideration when implementing the KNN algorithm, and a balance must be struck between model complexity and generalization performance. With the well-selected K value the identification situation is shown in the figure below:

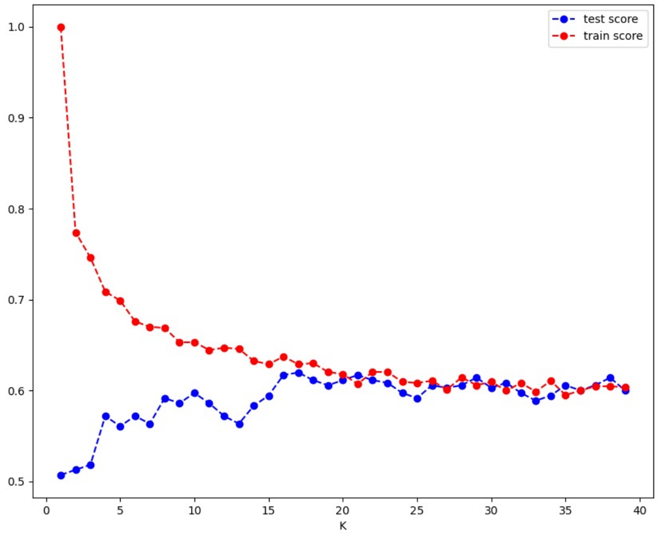


Fig. 9. KNN Accuracy with Different K value

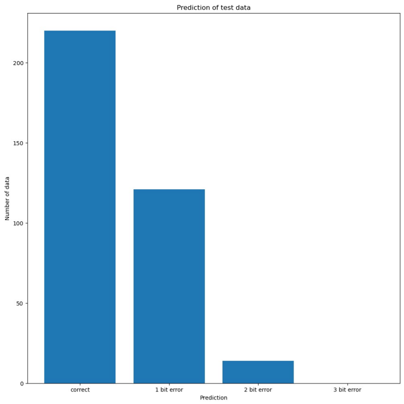


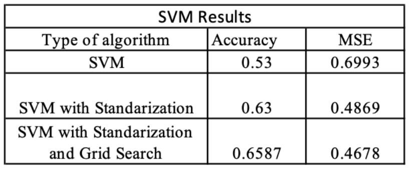
Fig. 10. KNN Identification Situation

B. SVM

In the SVM method, no cleaning was performed on the data, and after feeding the data into the SVM model fit, an accuracy of 53% was obtained. After cleaning the data, the accuracy improved to 63%. The default settings of the parameters of the SVM model were:

kernel='rbf', C=1.0, randomstate=0, degree=3, and gamma='auto'

A grid search was then performed for the hyperparameters, with C ranging from 0.1 to 10 and gamma ranging from 0.1 to 1, with three types of kernel functions, namely rbf, poly, and sigmoid. The best parameters for the grid search were found to be 'C': 2.3, 'gamma': 0.7, 'kernel': 'rbf'. The grid search was further refined, with C ranging from 1.8 to 2.8 and gamma ranging from 0.6 to 0.8. The final optimal hyperparameters were determined to be 'C': 2.326, 'gamma': 0.705, 'kernel': 'rbf', and the accuracy improved from 63% to 65.87%.



SVM Results		
Type of algorithm	Accuracy	MSE
SVM	0.53	0.6993
SVM with Standarization	0.63	0.4869
SVM with Standarization and Grid Search	0.6587	0.4678

Fig. 11. SVM Result

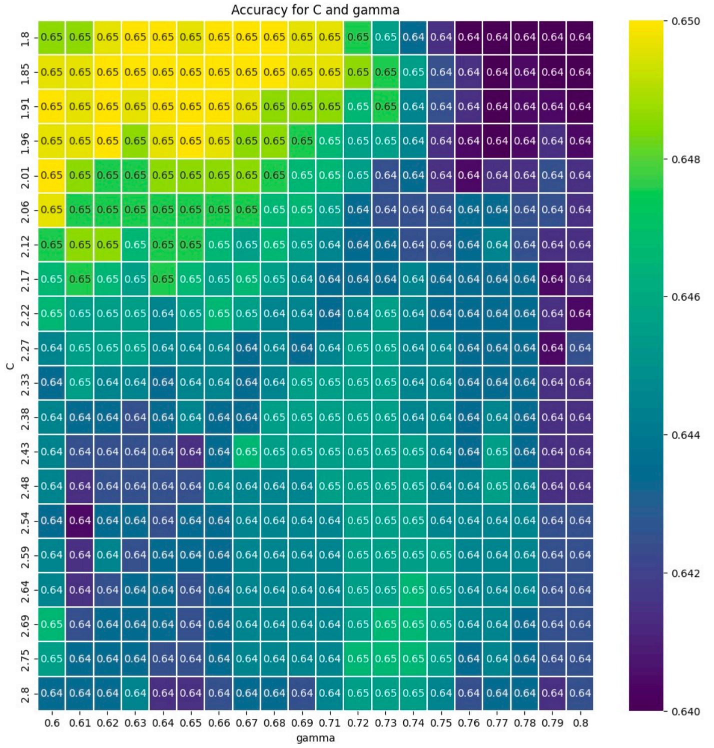


Fig. 12. SVM Grid Search Result

With the methods implementation, the MSE is decreasing, which is a good indicator, because MSE represents the situation of misidentification. The mis-identification figure is shown below:

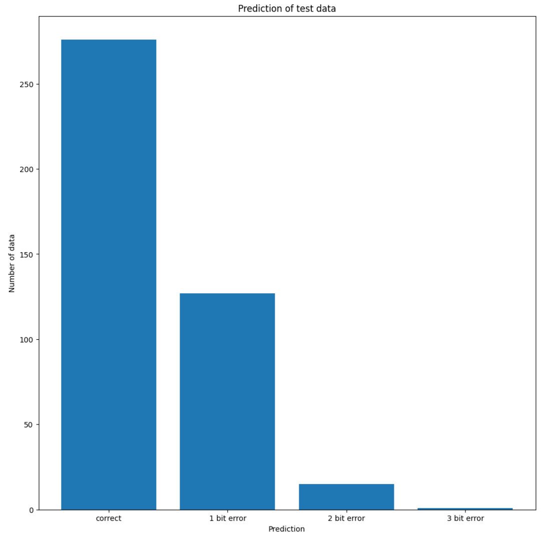
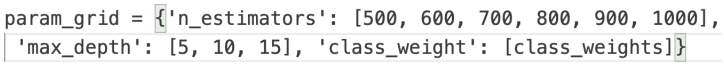


Fig. 13. SVM Prediction

C. Random Forest

The grid search parameters demonstrated as following: After trials and errors, the features with the most importance are citric acid and alcohol. The density of the wine still has some weight impact to the wine but not as much. The other variables are not as important. Even the consideration chlorides, free sulfur dioxide and total sulfur dioxide would disrupt the judgment of the algorithm and lower the accuracy of the classification.



```
param_grid = {'n_estimators': [500, 600, 700, 800, 900, 1000],
 'max_depth': [5, 10, 15], 'class_weight': [class_weights]}
```

Fig. 14. Random Forest Grid Search Code

(1: 10, 2: 10, 3: 69, 4: 1, 5: 1, 6: 1, 7: 1, 8: 34, 9: 10, 10: 10, 11: 80)

All method used the random state of 3 K-fold cross-validation is a technique used in machine learning to evaluate the performance of a model and to prevent overfitting. For cv = 10, K-fold average result is 0.68 which indicates the optimal accuracy does not deviate a lot from the average accuracy.

Fig. 15. Random Forest Result

The prediction correctness is shown in figure below:

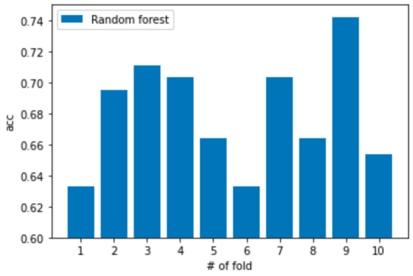


Fig. 16. Random Forest K-fold Analysis

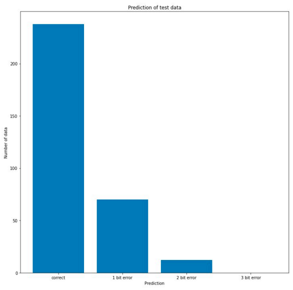


Fig. 17. Random Forest Correctness

IV. Conclusion

In this study, three machine learning classification methods, namely KNN, SVM, and Random Forest, were implemented to classify wines based on the UCI red-wine dataset. The KNN method had the lowest performance with an accuracy of 61.97% and a mean squared error (MSE) of 0.4985. The SVM method performed better than KNN, achieving an accuracy of 65.87% and an MSE of 0.4678. The Random Forest method outperformed the other two methods, with an accuracy of 75% and an MSE of 0.35.

The study also demonstrated that data cleaning and standardization were effective in improving the accuracy and lowering the MSE. These preprocessing techniques should be considered as a crucial step in machine learning classification tasks.

In conclusion, the Random Forest method was found to be the most effective in classifying wines based on the UCI red-wine dataset. The study highlights the importance of data cleaning and standardization in improving the accuracy and MSE of machine learning classification models.

References

[1] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, and J. Reis, ”Modeling wine preferences by data mining from physicochemical properties,” in IEEE Transactions on Neural Networks, vol. 17, no. 5, pp. 1136-1140, Sept. 2006. doi: 10.1109/TNN.2006.176

[2] Wine Quality Data Set, https://archive.ics.uci.edu/ml/datasets/wine+quality

[3] Scikit−learn Support Vector Machine (SVM), https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[4] Scikit−learn Random Forest Classifier, https://scikit-learn.org/stable/modules/generated sklearn.ensemble.RandomForestClassifier.html

[5] Scikit-learn Grid Search Cross Validation, https://scikit-learn.org/stable/modules/generated/sklearn.model selection.GridSearchCV.html

[6] Scikit−learn K−Nearest Neighbors Classifier, https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

In [238…

```python
# Data Cleaning and Analysis

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

import seaborn as sns
```

In [239…

```python
# read in the data
raw_data = pd.read_csv('winequality-red.csv', sep=';')
```

In [240…

```python
# remove repeated data
raw_data = raw_data.drop_duplicates()
raw_data.info()
raw_data.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1359 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1359 non-null   float64
 1   volatile acidity      1359 non-null   float64
 2   citric acid           1359 non-null   float64
 3   residual sugar        1359 non-null   float64
 4   chlorides             1359 non-null   float64
 5   free sulfur dioxide   1359 non-null   float64
 6   total sulfur dioxide  1359 non-null   float64
 7   density               1359 non-null   float64
 8   pH                    1359 non-null   float64
 9   sulphates             1359 non-null   float64
 10  alcohol               1359 non-null   float64
 11  quality               1359 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 138.0 KB
```

Out[240]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide
count	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000
mean	8.310596	0.529478	0.272333	2.523400	0.088124	15.893304
std	1.736990	0.183031	0.195537	1.352314	0.049377	10.447270
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000
75%	9.200000	0.640000	0.430000	2.600000	0.091000	21.000000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000

In [241…

```python
# remove outliers
for col in raw_data.columns:
    if col != 'quality':
        iqr = raw_data[col].quantile(0.75) - raw_data[col].quantile(0.25)
        upper_bound = raw_data[col].quantile(0.75) + 3 * iqr
        lower_bound = raw_data[col].quantile(0.25) - 3 * iqr
        raw_data = raw_data[(raw_data[col] < upper_bound) & (raw_data[col] >

raw_data.info()
raw_data.describe()
```

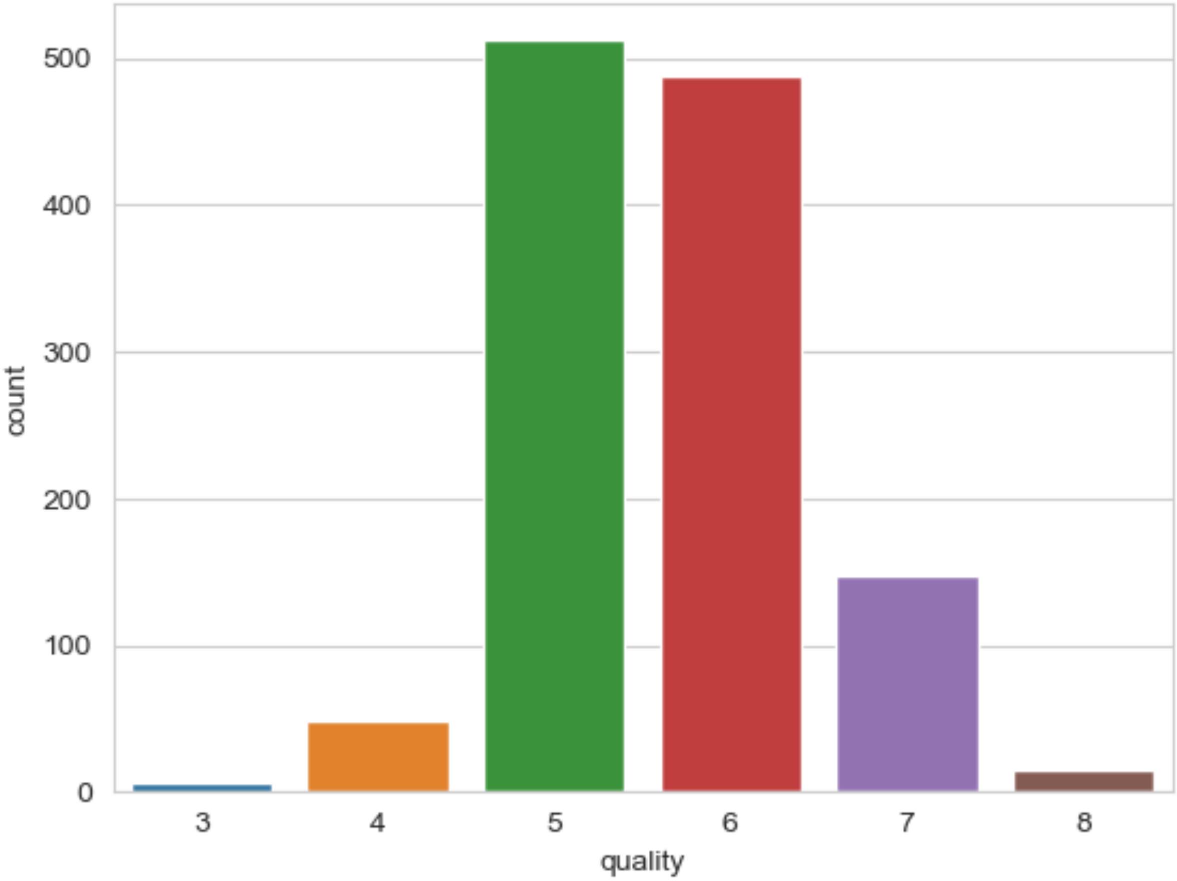
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1217 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1217 non-null   float64
 1   volatile acidity      1217 non-null   float64
 2   citric acid           1217 non-null   float64
 3   residual sugar        1217 non-null   float64
 4   chlorides             1217 non-null   float64
 5   free sulfur dioxide   1217 non-null   float64
 6   total sulfur dioxide  1217 non-null   float64
 7   density               1217 non-null   float64
 8   pH                    1217 non-null   float64
 9   sulphates             1217 non-null   float64
 10  alcohol               1217 non-null   float64
 11  quality               1217 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 123.6 KB
```

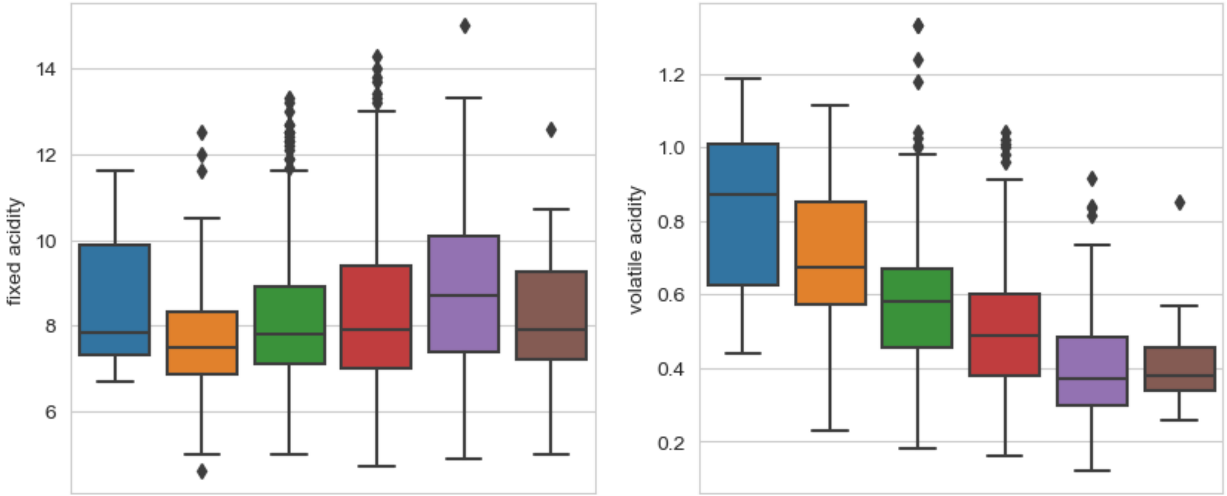
Out[241]:

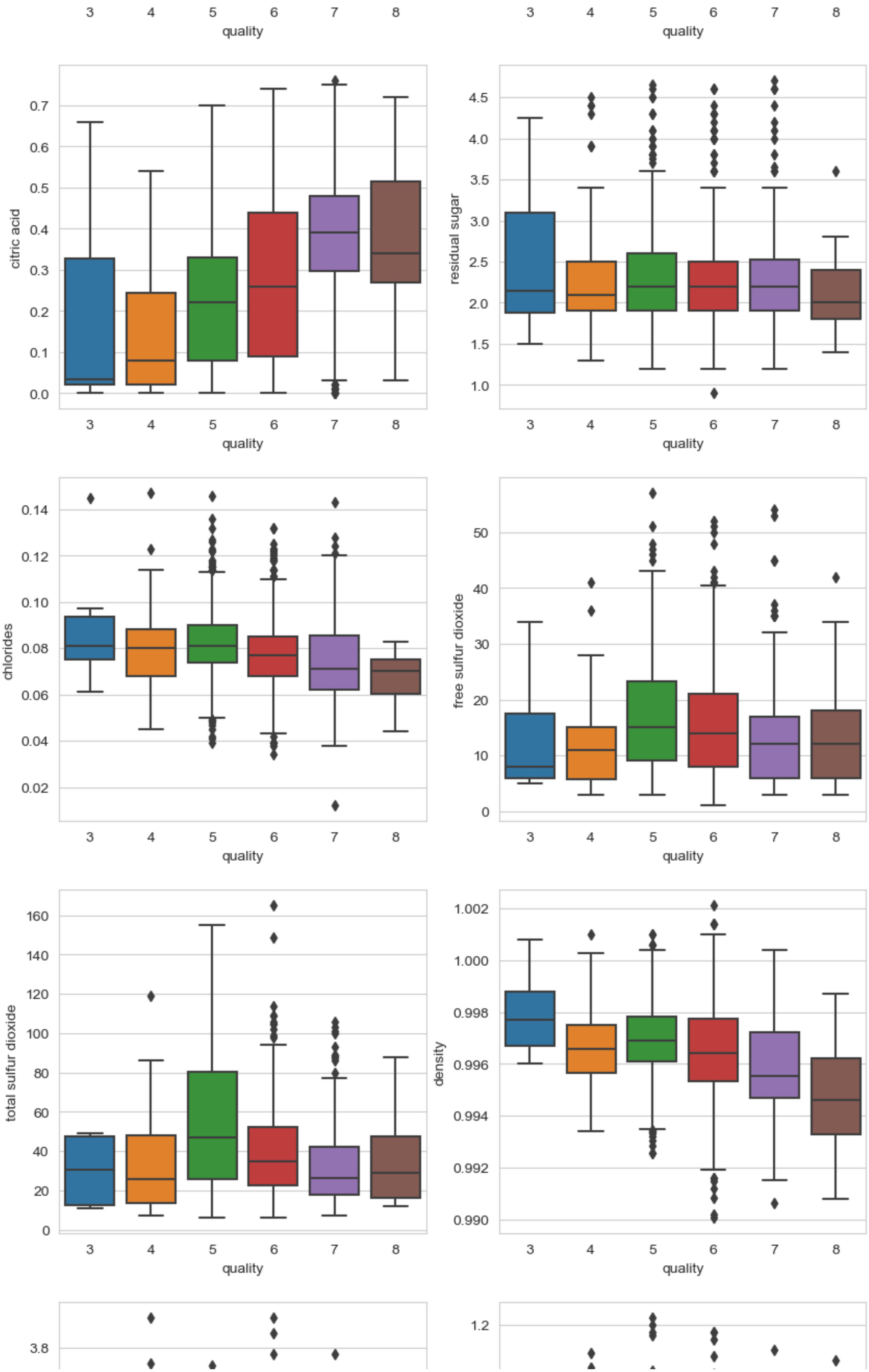
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total c
count	1217.000000	1217.000000	1217.000000	1217.000000	1217.000000	1217.000000	1217.0
mean	8.286360	0.526652	0.262021	2.278554	0.079132	15.759655	45.4
std	1.701773	0.180310	0.190105	0.593928	0.016570	9.894299	30.9
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.0
25%	7.100000	0.390000	0.090000	1.900000	0.069000	8.000000	22.0
50%	7.900000	0.520000	0.250000	2.200000	0.078000	14.000000	37.0
75%	9.200000	0.640000	0.420000	2.500000	0.088000	21.000000	60.0
max	15.000000	1.330000	0.760000	4.700000	0.147000	57.000000	165.0

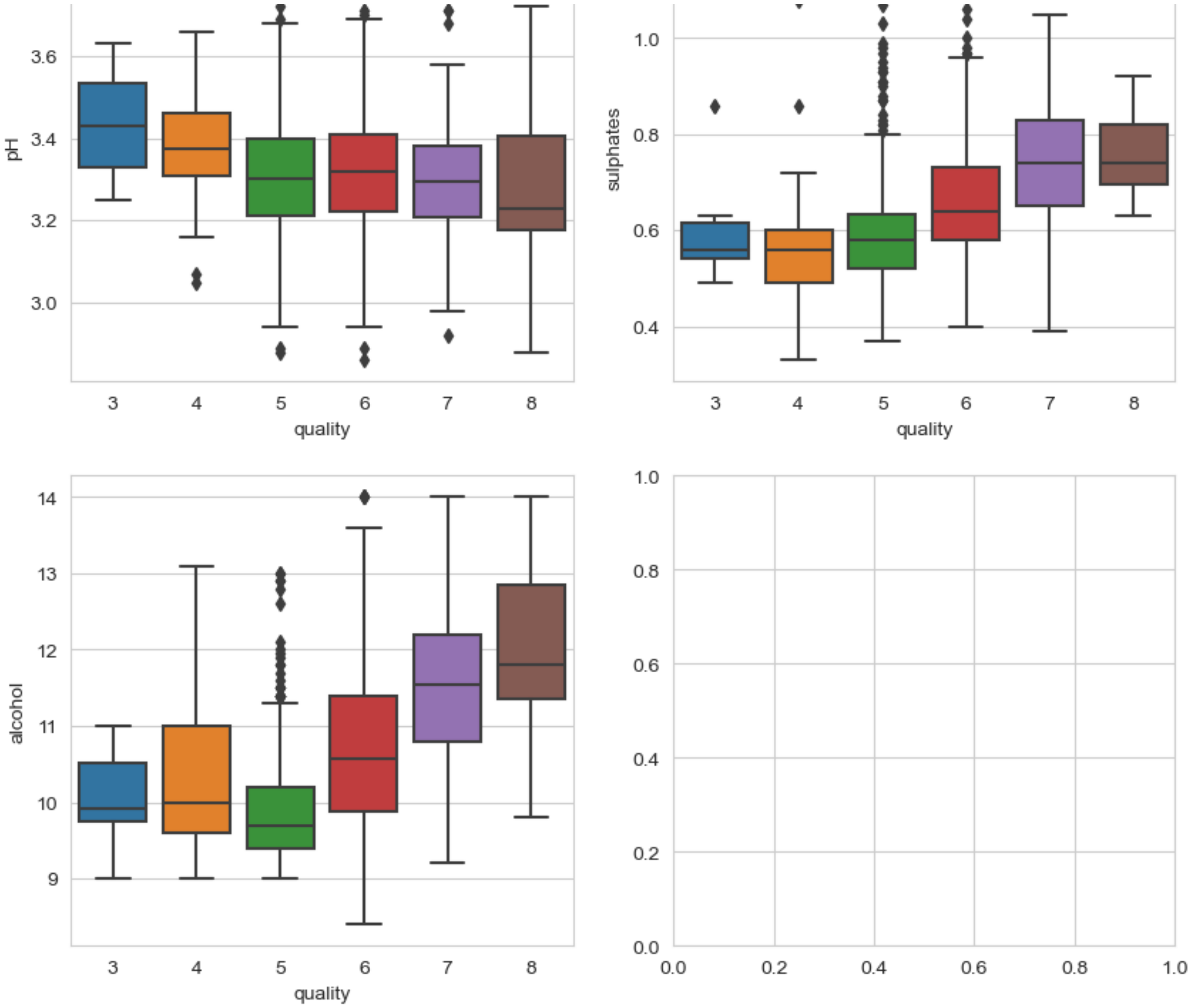
In [242…

```
# visualization
sns.countplot(x='quality', data=raw_data)

# box plot for each feature
fig, ax = plt.subplots(6, 2, figsize=(10, 30))
for i, col in enumerate(raw_data.columns):
    if col != 'quality':
        sns.boxplot(x='quality', y=col, data=raw_data, ax=ax[i//2][i%2])

plt.show()
# heatmap correlation matrix
fig = plt.figure(figsize=(10, 8))
corr = raw_data.corr()
sns.heatmap(corr, annot=True, cmap='BrBG')
```



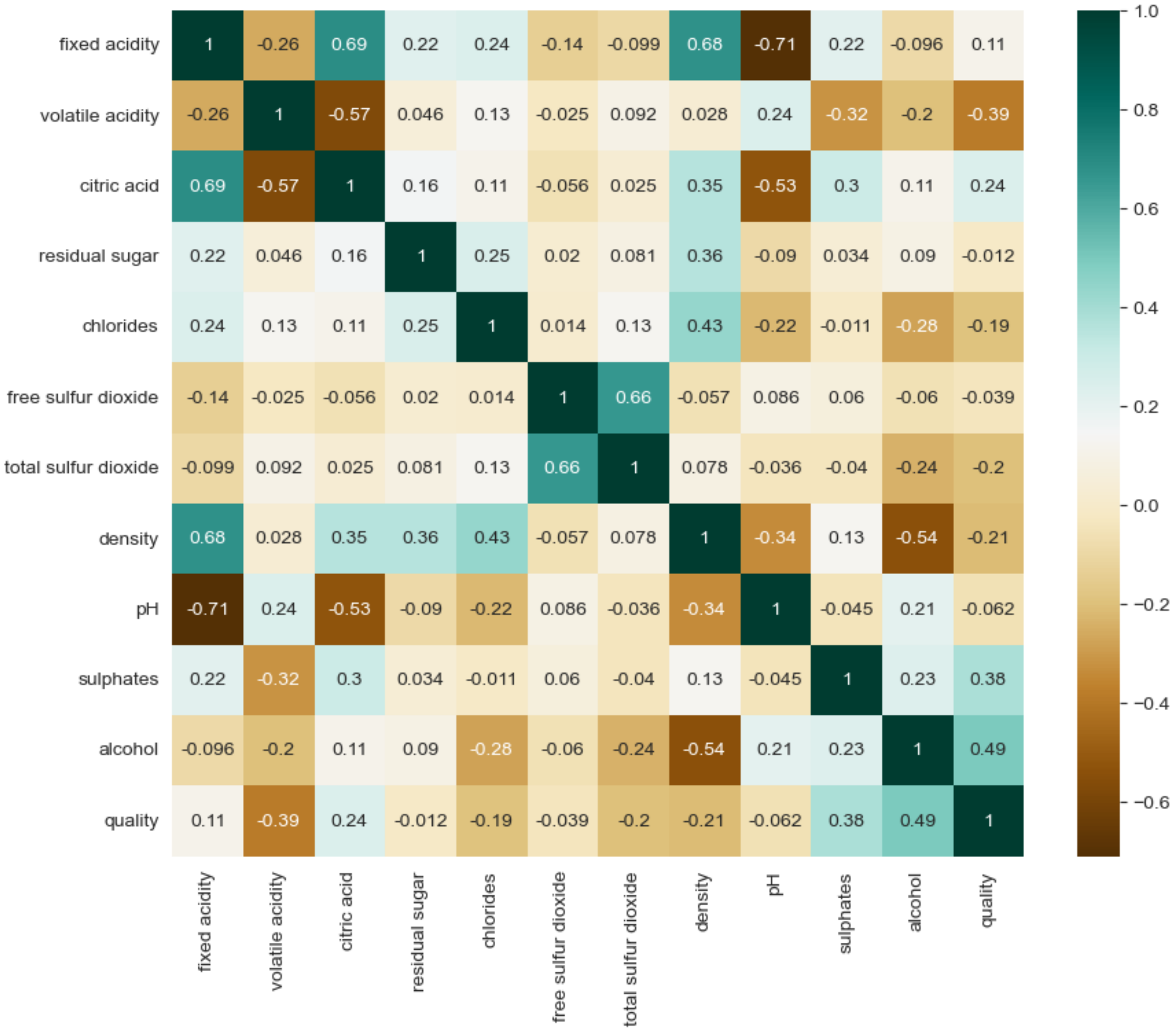



3 4 5 6 7 8
quality
3 4 5 6 7 8
quality
citric acid
0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7
3 4 5 6 7 8
quality
residual sugar
1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5
3 4 5 6 7 8
quality
chlorides
0.02 0.04 0.06 0.08 0.10 0.12 0.14
3 4 5 6 7 8
quality
free sulfur dioxide
0 10 20 30 40 50
3 4 5 6 7 8
quality
total sulfur dioxide
0 20 40 60 80 100 120 140 160
3 4 5 6 7 8
quality
density
0.990 0.992 0.994 0.996 0.998 1.000 1.002
3 4 5 6 7 8
quality
3.8
1.2



3.6
3.4
3.2
3.0
pH
3
4
5
6
7
8
quality
1.0
0.8
0.6
0.4
sulphates
3
4
5
6
7
8
quality
14
13
12
11
10
9
alcohol
3
4
5
6
7
8
quality
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

```
Out[242]: <AxesSubplot: >
```



In [243…

```
# normalize features except quality
from sklearn.preprocessing import StandardScaler
input_features = raw_data.drop('quality', axis=1)

scaler = StandardScaler()
input_features = scaler.fit_transform(input_features)

# contact quality to input features
processed_data = np.concatenate((input_features, raw_data['quality'].values.
```

In [244…

```python
# show processed data
processed_data = pd.DataFrame(processed_data, columns=raw_data.columns)
processed_data.info()
processed_data.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1217 entries, 0 to 1216
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1217 non-null   float64
 1   volatile acidity      1217 non-null   float64
 2   citric acid           1217 non-null   float64
 3   residual sugar        1217 non-null   float64
 4   chlorides             1217 non-null   float64
 5   free sulfur dioxide   1217 non-null   float64
 6   total sulfur dioxide  1217 non-null   float64
 7   density               1217 non-null   float64
 8   pH                    1217 non-null   float64
 9   sulphates             1217 non-null   float64
 10  alcohol               1217 non-null   float64
 11  quality               1217 non-null   float64
dtypes: float64(12)
memory usage: 114.2 KB
```

Out[244]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free d
count	1.217000e+03	1.217000e+03	1217.000000	1.217000e+03	1.217000e+03	1.21700
mean	-2.802469e-16	-4.028550e-16	0.000000	-1.634774e-16	-4.437243e-16	-5.2546
std	1.000411e+00	1.000411e+00	1.000411	1.000411e+00	1.000411e+00	1.00041
min	-2.167078e+00	-2.256216e+00	-1.378863	-2.322031e+00	-4.053072e+00	-1.49234
25%	-6.974183e-01	-7.581809e-01	-0.905247	-6.376348e-01	-6.117312e-01	-7.8457
50%	-2.271271e-01	-3.690493e-02	-0.063261	-1.323158e-01	-6.836149e-02	-1.77918
75%	5.370962e-01	6.288883e-01	0.831348	3.730032e-01	5.353826e-01	5.2985
max	3.946708e+00	4.457199e+00	2.620567	4.078676e+00	4.097473e+00	4.16980

In [245…

```python
print(raw_data['quality'].value_counts())
```

```
5    512
6    488
7    148
4     48
8     15
3      6
Name: quality, dtype: int64
```

In [246…

```python
# save processed data to csv
processed_data.to_csv('processed_data.csv', index=False)
```

In [39]:

```
# KNN

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

from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
```

In [40]:

```
# read the data
raw_data = pd.read_csv('winequality-red.csv', sep=';')
```

In [41]:

```
# remove outliers
for col in raw_data.columns:
        if col != 'quality':
                iqr = raw_data[col].quantile(0.75) - raw_data[col].quantile(
                upper_bound = raw_data[col].quantile(0.75) + 2.5 * iqr
                lower_bound = raw_data[col].quantile(0.25) - 2.5 * iqr
                raw_data = raw_data[(raw_data[col] < upper_bound) & (raw_dat

# save the cleaned data
raw_data.to_csv('cleaned_data.csv', index=False)
```

In [42]:

```
# raw_data.info()
# remove repeated data
raw_data = raw_data.drop_duplicates()
raw_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1182 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1182 non-null   float64
 1   volatile acidity      1182 non-null   float64
 2   citric acid           1182 non-null   float64
 3   residual sugar        1182 non-null   float64
 4   chlorides             1182 non-null   float64
 5   free sulfur dioxide   1182 non-null   float64
 6   total sulfur dioxide  1182 non-null   float64
 7   density               1182 non-null   float64
 8   pH                    1182 non-null   float64
 9   sulphates             1182 non-null   float64
 10  alcohol               1182 non-null   float64
 11  quality               1182 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 120.0 KB
```

In [43]:

```
raw_data.describe()
```

Out[43]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	tota
count	1182.000000	1182.000000	1182.000000	1182.000000	1182.000000	1182.000000	1182.
mean	8.268613	0.525816	0.260398	2.255118	0.078864	15.692893	45
std	1.674834	0.176989	0.189768	0.545431	0.015813	9.689748	30
min	4.700000	0.120000	0.000000	0.900000	0.038000	1.000000	6.
25%	7.100000	0.390000	0.090000	1.900000	0.069000	8.000000	22.
50%	7.900000	0.520000	0.245000	2.200000	0.078000	14.000000	37.
75%	9.200000	0.640000	0.410000	2.500000	0.088000	21.000000	60.
max	14.300000	1.240000	0.760000	4.300000	0.136000	53.000000	152.

In [44]:

```
# Standardization
scaler = StandardScaler()
raw_data.iloc[:, :-1] = scaler.fit_transform(raw_data.iloc[:, :-1])
```

In [45]:

```
# train test split
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(raw_data.iloc[:, :-1], r
```

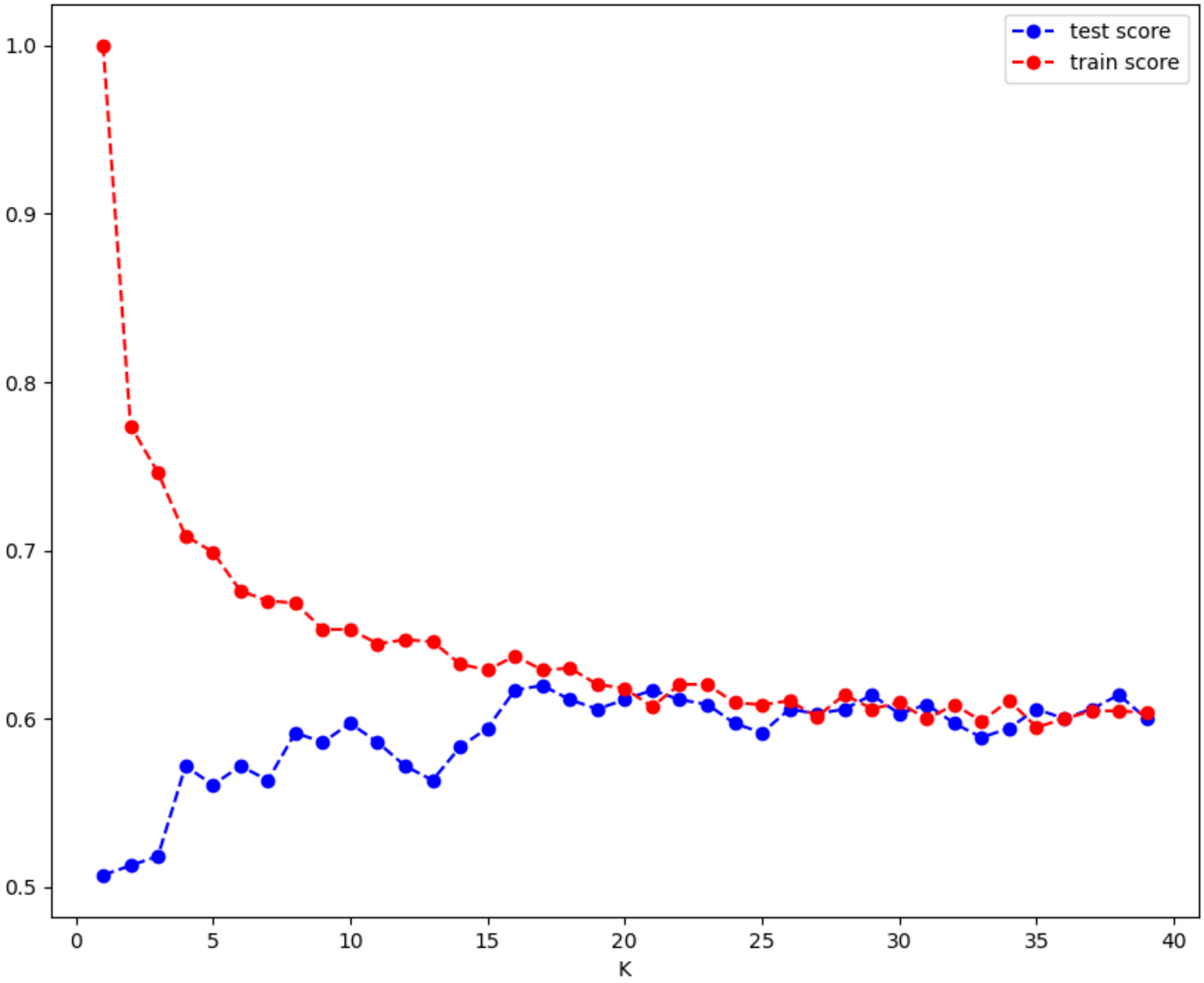
In [46]:

```
# # gird search on K and distance function
# from sklearn.model_selection import GridSearchCV
# k_range = list(range(1, 50))
# weight_options = ['uniform', 'distance']
# distance_metric = ['euclidean', 'manhattan', 'minkowski', 'cosine']
# param_grid = dict(n_neighbors=k_range, weights=weight_options, metric=dist
# knn = KNeighborsClassifier()
# grid = GridSearchCV(knn, param_grid, cv=10, scoring='accuracy', return_tra
# grid.fit(X_train, y_train)
#
# # print the best parameters
# print(grid.best_params_)
# print(grid.best_score_)
```

In [47]:

```
train_score = []
test_score = []
for i in range(1, 40):

        knn = KNeighborsClassifier(n_neighbors = i)
        knn.fit(X_train, y_train)
        pred_i = knn.predict(X_test)
        test_score.append(knn.score(X_test, y_test))
        train_score.append(knn.score(X_train, y_train))


plt.figure(figsize =(10, 8))

plt.plot(range(1, 40), test_score, color ='blue', linestyle ='dashed', marke


plt.plot(range(1, 40), train_score, color ='red', linestyle ='dashed', marke

plt.xlabel('K')
plt.legend(['test score', 'train score'], loc ='upper right')

print("Maximum test score:",max(test_score),"at K =",test_score.index(max(te
```

```
Maximum test score: 0.6197183098591549 at K = 17
```



In [48]:

```
# k = 17
classifier = KNeighborsClassifier(n_neighbors = 17)
classifier.fit(X_train,y_train)
```

Out[48]:

```
▾        KNeighborsClassifier
KNeighborsClassifier(n_neighbors=17)
```

In [49]:

```
#Predicting the ouput from input data (x_train) and (y_train)
y_pred1 = classifier.predict(X_train)
y_pred2 = classifier.predict(X_test)
```

In [50]:

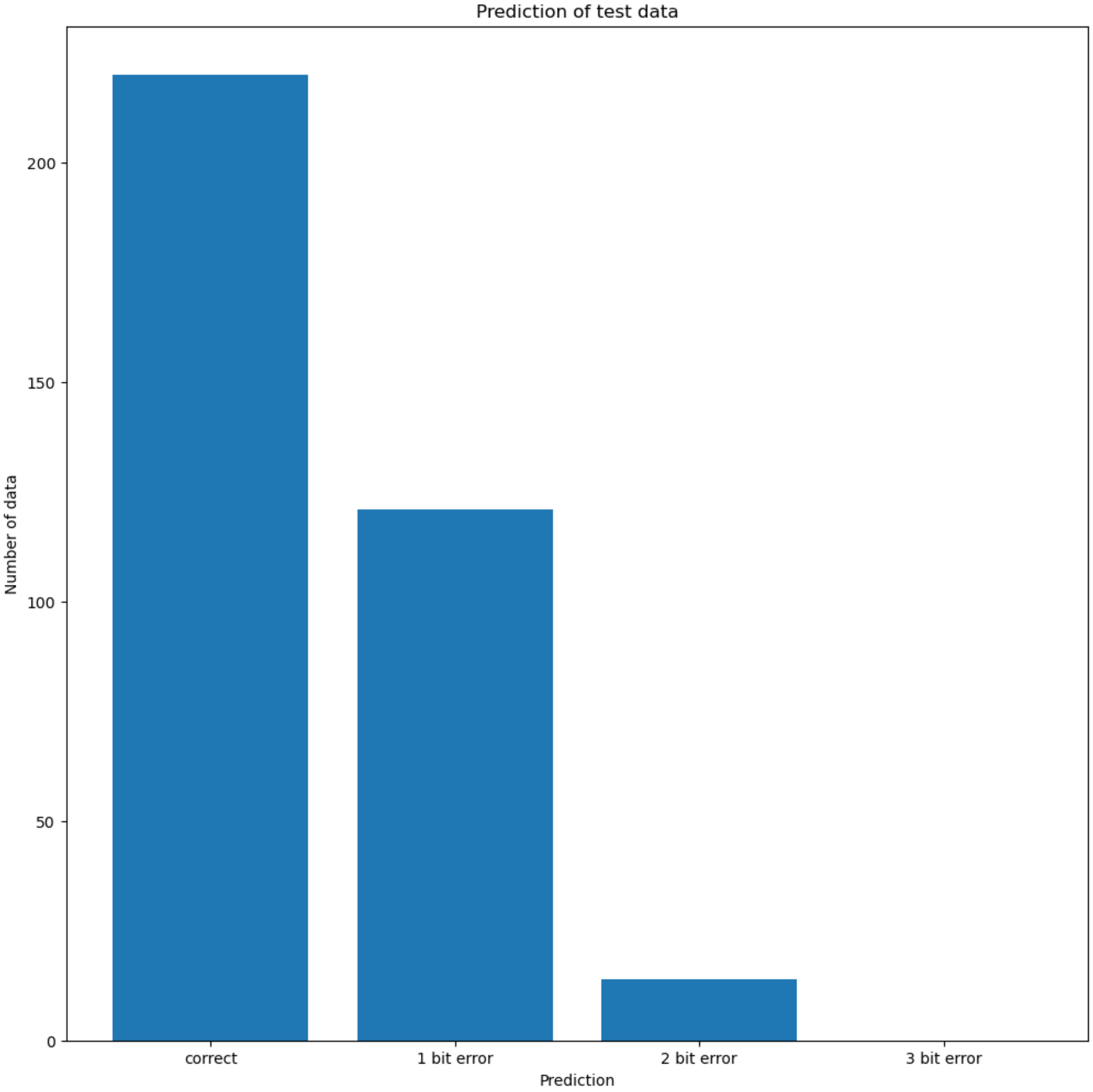
```
from sklearn.metrics import accuracy_score, mean_squared_error

print("train score",accuracy_score(y_train, y_pred1))
print("test score",accuracy_score(y_test, y_pred2))
print("MSE",mean_squared_error(y_test, y_pred2))

y_test = np.array(y_test)
```

```
train score 0.6287787182587666
test score 0.6197183098591549
MSE 0.49859154929577465
```

In [51]:

```
# visualization
correct = 0
one_bit_error = 0
two_bit_error = 0
threemore_bit_error = 0
print('Shap of y_pred: ', y_test.shape)
for i in range(len(y_pred2)):
        if y_pred2[i] == y_test[i]:
                correct += 1
        elif abs(y_pred2[i] - y_test[i]) == 1:
                one_bit_error += 1
        elif abs(y_pred2[i] - y_test[i]) == 2:
                two_bit_error += 1
        else:
                threemore_bit_error += 1
plt.figure(figsize=(12, 12))
plt.bar(['correct', '1 bit error', '2 bit error', '3 bit error'], [correct,
plt.title('Prediction of test data')
plt.xlabel('Prediction')
plt.ylabel('Number of data')
plt.show()
```

```
Shap of y_pred:  (355,)
```


Prediction of test data
Number of data
200
150
100
50
0
correct
1 bit error
2 bit error
3 bit error
Prediction

In [51]:

SVM

May 5, 2023

[ ]:

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import tqdm


# Load data from winequality-red.csv
Data = np.loadtxt('winequality-red.csv', delimiter=';', skiprows=1)
Data = np.loadtxt('cleaned_data.csv', delimiter=',', skiprows=1)
print(Data.shape)
y = Data[:, -1]
X = Data[:, :-1]


# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.
 ↪3,random_state=70)
```

```
(1395, 12)
```

[ ]:

```
svc_clf = SVC(kernel='rbf', C=1.0, random_state=1, degree=3, gamma='auto')
svc_clf.fit(X_train,y_train)
y_pred = svc_clf.predict(X_test)
print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.53
```

[ ]:

```
# Calculate the MSE
MSE = np.mean((y_pred - y_test)**2)
print('MSE: %.4f' % MSE)
```

```
MSE: 0.6993
```

[ ]:

```
# Standardize the features
sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)
svc_clf.fit(X_train_std,y_train)
y_pred = svc_clf.predict(X_test_std)
print('Accuracy: %.2f' % accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.63
```

[ ]:

```
# Calculate the MSE
MSE = np.mean((y_pred - y_test)**2)
print('MSE: %.4f' % MSE)
```

```
MSE: 0.4869
```

[ ]:

```
# Grid search for best parameters
param_grid = {'C': np.linspace(0.1, 10, 10),
                'gamma': np.linspace(0.1, 1, 10),
                'kernel': ['rbf', 'poly', 'sigmoid']}
grid = GridSearchCV(SVC(), param_grid, refit=True, verbose=3)
grid.fit(X_train_std, y_train)
print(grid.best_params_)
print(grid.best_estimator_)
grid_predictions = grid.predict(X_test_std)
print('Accuracy: %.4f' % accuracy_score(y_test, grid_predictions))
```

```
Fitting 5 folds for each of 300 candidates, totalling 1500 fits
[CV 1/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.612 total time=   0.0s
[CV 2/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.579 total time=   0.0s
[CV 3/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.641 total time=   0.0s
[CV 4/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.564 total time=   0.0s
[CV 5/5] END …C=0.1, gamma=0.1, kernel=rbf;, score=0.610 total time=   0.0s
[CV 1/5] END …C=0.1, gamma=0.1, kernel=poly;, score=0.566 total time=   0.0s
[CV 2/5] END …C=0.1, gamma=0.1, kernel=poly;, score=0.590 total time=   0.0s
[CV 3/5] END …C=0.1, gamma=0.1, kernel=poly;, score=0.615 total time=   0.0s
```

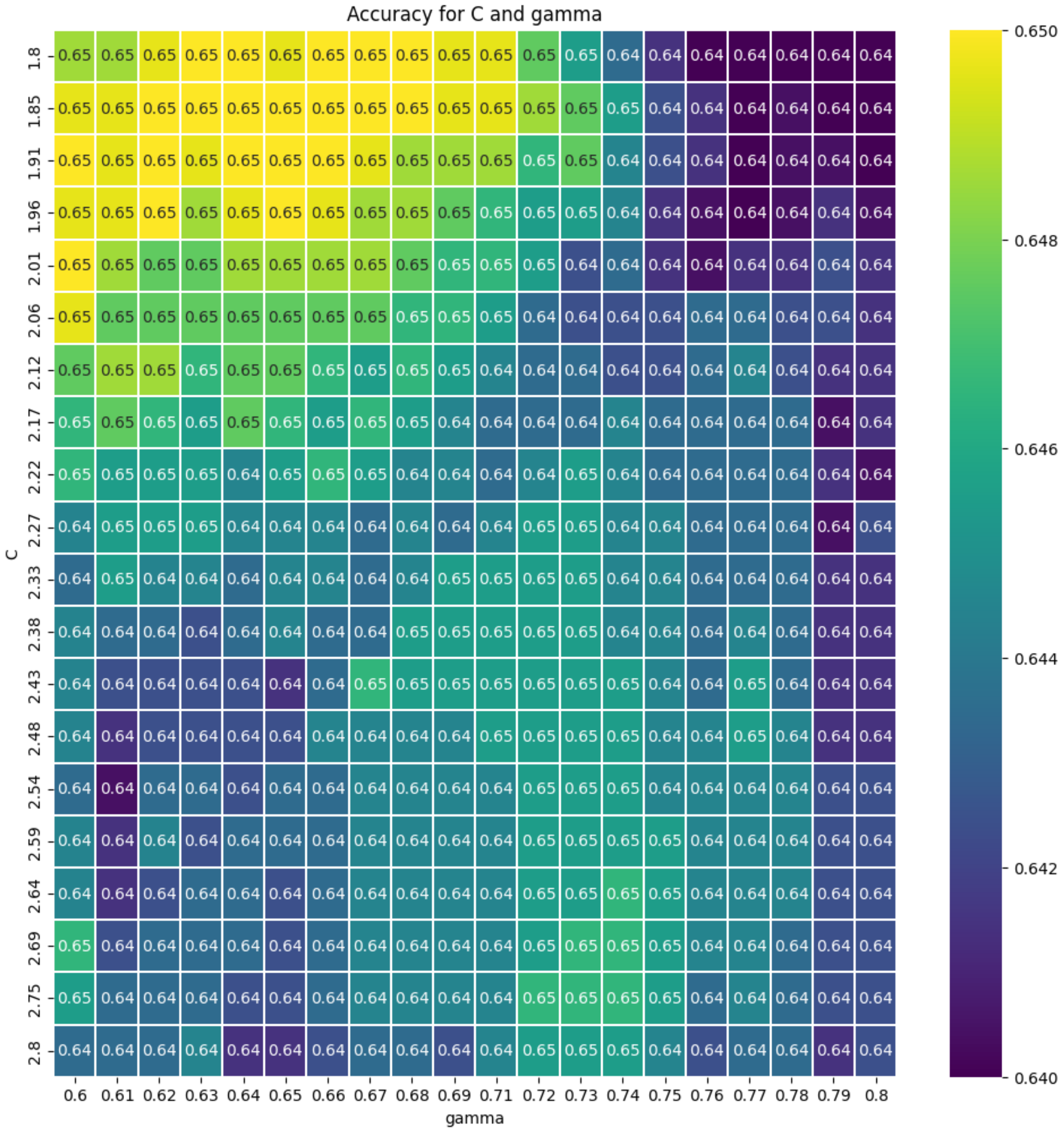
```
c:\Users\quanh\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\model_selection\_split.py:700: UserWarning: The least populated
class in y has only 3 members, which is less than n_splits=5.
  warnings.warn(
```

```
[CV 4/5] END …C=0.1, gamma=0.1, kernel=poly;, score=0.574 total time=   0.0s
[CV 5/5] END …C=0.1, gamma=0.1, kernel=poly;, score=0.569 total time=   0.0s
[CV 1/5] END ..C=0.1, gamma=0.1, kernel=sigmoid;, score=0.597 total time=   0.0s
[CV 2/5] END ..C=0.1, gamma=0.1, kernel=sigmoid;, score=0.600 total time=   0.0s
[CV 3/5] END ..C=0.1, gamma=0.1, kernel=sigmoid;, score=0.615 total time=   0.0s
[CV 4/5] END ..C=0.1, gamma=0.1, kernel=sigmoid;, score=0.569 total time=   0.0s
```

```
[CV 5/5] END C=2.8, gamma=0.768421052631579, kernel=rbf;, score=0.656 total
time=   0.0s
[CV 1/5] END C=2.8, gamma=0.7789473684210526, kernel=rbf;, score=0.612 total
time=   0.0s
[CV 2/5] END C=2.8, gamma=0.7789473684210526, kernel=rbf;, score=0.605 total
time=   0.0s
[CV 3/5] END C=2.8, gamma=0.7789473684210526, kernel=rbf;, score=0.677 total
time=   0.0s
[CV 4/5] END C=2.8, gamma=0.7789473684210526, kernel=rbf;, score=0.667 total
time=   0.0s
[CV 5/5] END C=2.8, gamma=0.7789473684210526, kernel=rbf;, score=0.656 total
time=   0.0s
[CV 1/5] END C=2.8, gamma=0.7894736842105263, kernel=rbf;, score=0.612 total
time=   0.0s
[CV 2/5] END C=2.8, gamma=0.7894736842105263, kernel=rbf;, score=0.600 total
time=   0.0s
[CV 3/5] END C=2.8, gamma=0.7894736842105263, kernel=rbf;, score=0.667 total
time=   0.0s
[CV 4/5] END C=2.8, gamma=0.7894736842105263, kernel=rbf;, score=0.667 total
time=   0.0s
[CV 5/5] END C=2.8, gamma=0.7894736842105263, kernel=rbf;, score=0.662 total
time=   0.0s
[CV 1/5] END …C=2.8, gamma=0.8, kernel=rbf;, score=0.607 total time=   0.0s
[CV 2/5] END …C=2.8, gamma=0.8, kernel=rbf;, score=0.600 total time=   0.0s
[CV 3/5] END …C=2.8, gamma=0.8, kernel=rbf;, score=0.672 total time=   0.0s
[CV 4/5] END …C=2.8, gamma=0.8, kernel=rbf;, score=0.672 total time=   0.0s
[CV 5/5] END …C=2.8, gamma=0.8, kernel=rbf;, score=0.662 total time=   0.0s
{'C': 1.8526315789473684, 'gamma': 0.631578947368421, 'kernel': 'rbf'}
SVC(C=1.8526315789473684, gamma=0.631578947368421)
Accuracy: 0.6587
```

[ ]:

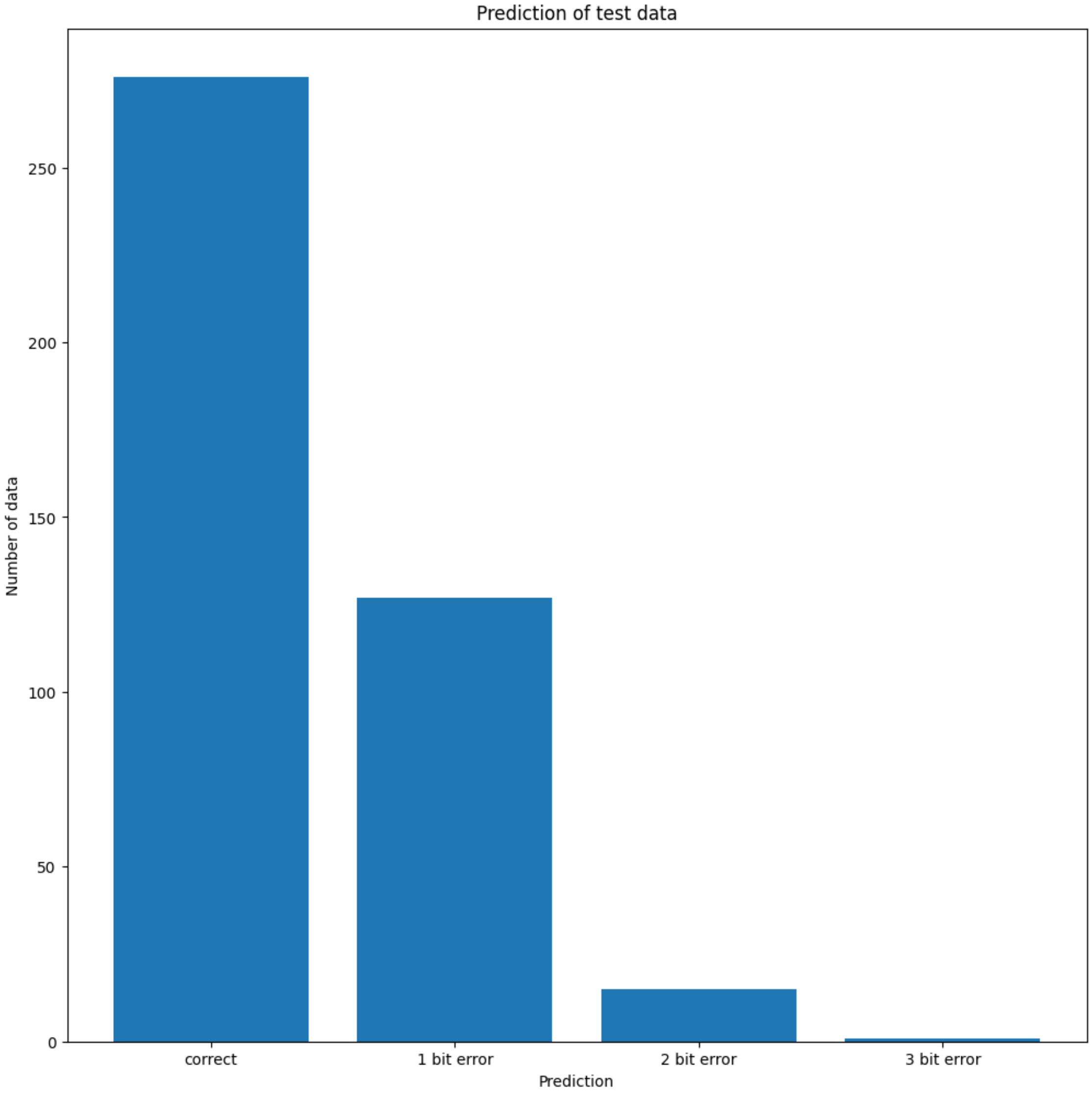
```
# visualize the accuracy of the model with with best grid search parameters
C_range = np.linspace(1.8, 2.8, 20)
gamma_range = np.linspace(0.6, 0.8, 20)
scores = grid.cv_results_['mean_test_score'].reshape(len(C_range),␣
 ↪len(gamma_range))
plt.figure(figsize=(12, 12))
sns.heatmap(scores, vmin=0.64, vmax=0.65, cmap='viridis', annot=True,␣
 ↪annot_kws={"size": 10}, fmt='.2f', linewidths=0.01,
            linecolor="white", xticklabels=np.round(gamma_range, 2),␣
 ↪yticklabels=np.round(C_range, 2))
plt.title('Accuracy for C and gamma')
plt.xlabel('gamma')
plt.ylabel('C')
plt.show()
```



[ ]:

```
# visualize the prediction of model of test, make them in 3 plots
# bar plot of correct prediction, 1 bit error prediction, 2 bit error prediction, 3 bit error prediction
# best parameters: C =1.8526315789473684, gamma=0.631578947368421
svc_clf = SVC(kernel='rbf', C=1.8526315789473684, random_state=1, degree=3, gamma=0.631578947368421)
svc_clf.fit(X_train_std,y_train)
y_pred = svc_clf.predict(X_test_std)
correct = 0
one_bit_error = 0
two_bit_error = 0
```

```
threemore_bit_error = 0
for i in range(len(y_pred)):
    if y_pred[i] == y_test[i]:
        correct += 1
    elif abs(y_pred[i] - y_test[i]) == 1:
        one_bit_error += 1
    elif abs(y_pred[i] - y_test[i]) == 2:
        two_bit_error += 1
    else:
        threemore_bit_error += 1
plt.figure(figsize=(12, 12))
plt.bar(['correct', '1 bit error', '2 bit error', '3 bit error'], [correct,␣
 ↪one_bit_error, two_bit_error, threemore_bit_error])
plt.title('Prediction of test data')
plt.xlabel('Prediction')
plt.ylabel('Number of data')
plt.show()

#print MSE of prediction
print('MSE of prediction: %.4f' % np.mean((y_pred - y_test)**2))
```



```
MSE of prediction: 0.4678
```

ESE417_final_ANN_AQ

May 5, 2023

[ ]:

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

# load red wine data
data = np.loadtxt('winequality-red.csv', delimiter=';', skiprows=1)
X = data[:, :-1]
y = data[:, -1]


# split data into training and testing sets
from sklearn.model_selection import train_test_split
tr_X, te_X, tr_Y, te_Y = train_test_split(X, y, test_size=0.2, random_state=5)
```

Random Forest Method

[ ]:

```
#random forest method classify wine quality
import tqdm
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

#use tqdm to show the progress of the training
clf = RandomForestClassifier(n_estimators=1000, max_depth=10, random_state=3)
clf.fit(tr_X, tr_Y)
pred_Y = clf.predict(te_X)

#show the classification report
print(classification_report(te_Y, pred_Y))

#show the MSE
from sklearn.metrics import mean_squared_error
print("Mean squared error: %.2f" % mean_squared_error(te_Y, pred_Y))
```

```
              precision    recall  f1-score   support

         3.0       0.00      0.00      0.00         1
         4.0       0.00      0.00      0.00         6
         5.0       0.86      0.83      0.84       152
         6.0       0.63      0.83      0.72       115
         7.0       0.80      0.40      0.53        40
         8.0       0.00      0.00      0.00         6

    accuracy                           0.74       320
   macro avg       0.38      0.34      0.35       320
weighted avg       0.73      0.74      0.72       320

Mean squared error: 0.37
```

```
/Users/anranqiao/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/anranqiao/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/anranqiao/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[ ]:

```python
# weight the features
#generate class weight
from sklearn.utils import class_weight

class_weights = { 1:10,2:10,3:69, 4:1, 5:1, 6:1, 7:1, 8:34, 9:10, 10:10, 11:80}
clf = RandomForestClassifier(n_estimators=1000, max_depth=10, random_state=3,␣
 ↪class_weight=class_weights)

clf.fit(tr_X, tr_Y)
pred_Y = clf.predict(te_X)

#show the classification report
print(classification_report(te_Y, pred_Y))

#print mean square error

print("Mean squared error: %.2f" % mean_squared_error(te_Y, pred_Y))
```

```
              precision    recall  f1-score   support

         3.0       0.00      0.00      0.00         1
         4.0       0.00      0.00      0.00         6
         5.0       0.86      0.84      0.85       152
         6.0       0.63      0.84      0.72       115
         7.0       0.82      0.35      0.49        40
         8.0       1.00      0.17      0.29         6

    accuracy                           0.75       320
   macro avg       0.55      0.37      0.39       320
weighted avg       0.76      0.75      0.73       320

Mean squared error: 0.36
```

```
/Users/anranqiao/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/anranqiao/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/anranqiao/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[ ]:

```
#grid search to find the best parameters
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier

#use grid search to find the best parameters

param_grid = {'n_estimators': [500, 600, 700, 800, 900, 1000], 'max_depth': [5,␣
 ↪10, 15], 'class_weight': [class_weights]}
clf = GridSearchCV(RandomForestClassifier(), param_grid, cv=5,␣
 ↪scoring='accuracy')
clf.fit(tr_X, tr_Y)
print("Best parameters set found on development set:")
print(clf.best_params_)
print("Grid scores on development set:")
means = clf.cv_results_['mean_test_score']
stds = clf.cv_results_['std_test_score']
#show the classification report

pred_Y = clf.predict(te_X)
print(classification_report(te_Y, pred_Y))
print("Mean squared error: %.2f" % mean_squared_error(te_Y, pred_Y))
```

```
Best parameters set found on development set:
{'class_weight': {1: 10, 2: 10, 3: 69, 4: 1, 5: 1, 6: 1, 7: 1, 8: 34, 9: 10, 10:
10, 11: 80}, 'max_depth': 15, 'n_estimators': 500}
Grid scores on development set:
              precision    recall  f1-score   support

         3.0       0.00      0.00      0.00         1
         4.0       0.00      0.00      0.00         6
         5.0       0.86      0.83      0.85       152
         6.0       0.64      0.83      0.72       115
         7.0       0.74      0.42      0.54        40
         8.0       1.00      0.17      0.29         6

    accuracy                           0.75       320
   macro avg       0.54      0.37      0.40       320
weighted avg       0.75      0.75      0.73       320

Mean squared error: 0.35
```

```
/Users/anranqiao/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/anranqiao/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Users/anranqiao/anaconda3/lib/python3.9/site-
packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

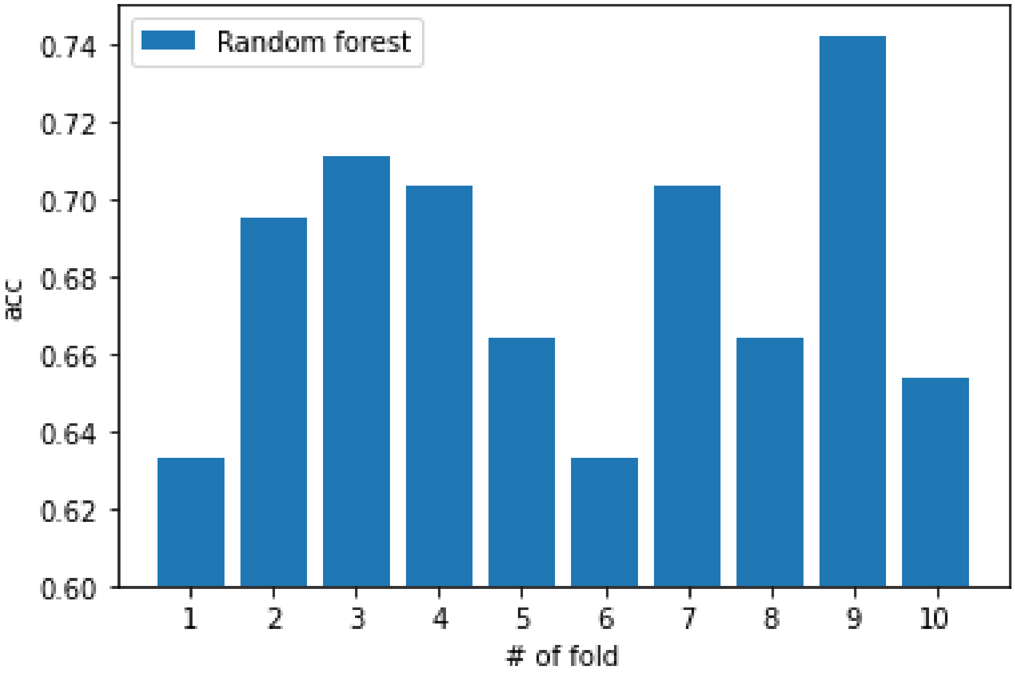
[ ]:

```python
#show the feature importance
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score

model_s = cross_val_score(clf, tr_X, tr_Y, cv=10)
#print the cross validation score
print("Cross validation score: %.2f" % model_s.mean())

#show the feature importance
plt.bar(range(1, 11), model_s, label='Random forest')
plt.xticks(range(1, 11))
plt.ylim(0.6, 0.75)
plt.legend()
plt.xlabel("# of fold")
plt.ylabel("acc")
plt.savefig("fig6.png", transparancy=True)
```

```
/Users/anranqiao/anaconda3/lib/python3.9/site-
packages/sklearn/model_selection/_split.py:700: UserWarning: The least populated
class in y has only 9 members, which is less than n_splits=10.
  warnings.warn(
```

```
Cross validation score: 0.68
```

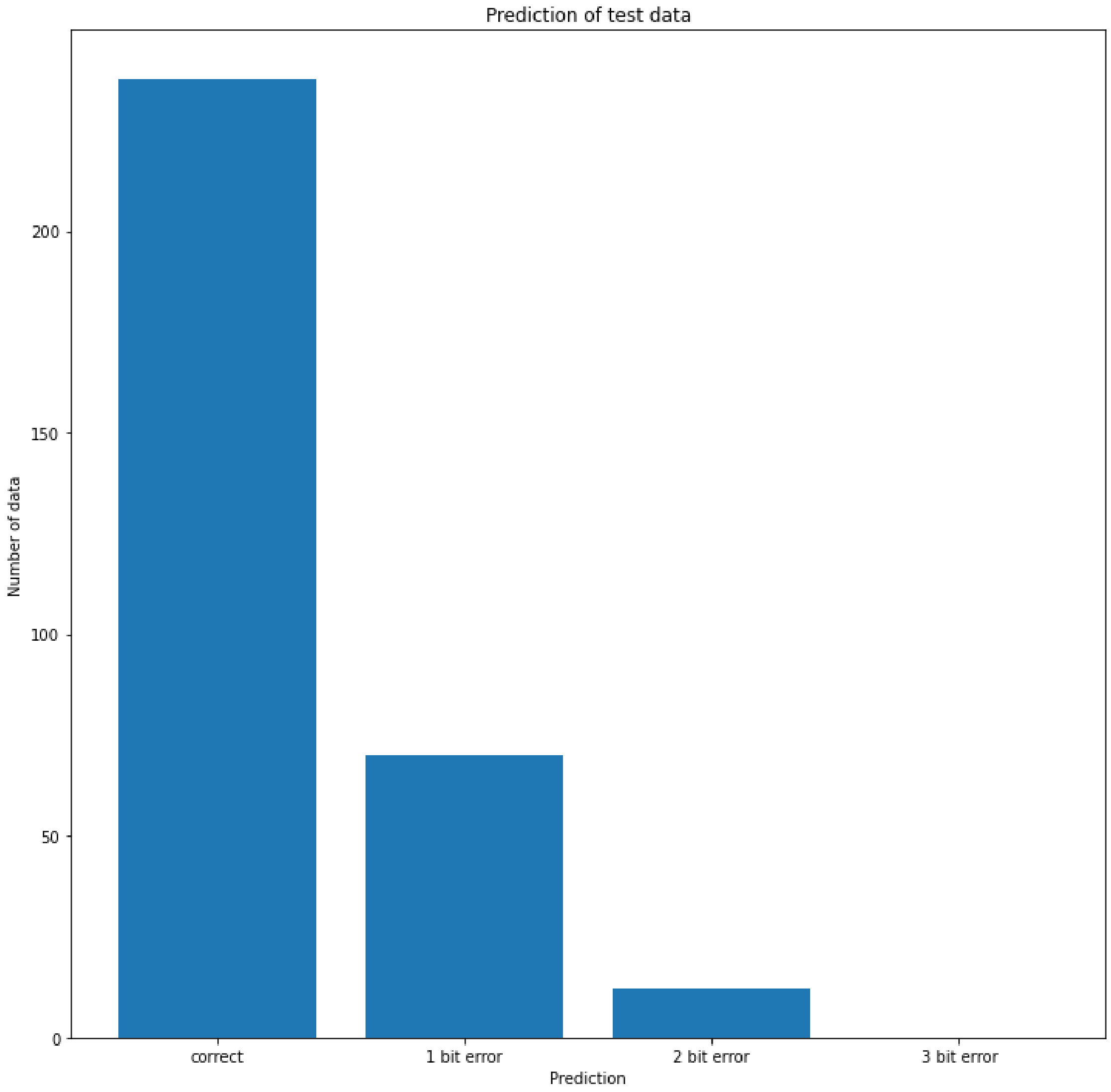
```
/var/folders/jb/4h3n8x816qv1wgp3qpxgrv_40000gn/T/ipykernel_4504/3455214418.py:16
: MatplotlibDeprecationWarning: savefig() got unexpected keyword argument
"transparancy" which is no longer supported as of 3.3 and will become an error
in 3.6
  plt.savefig("fig6.png", transparancy=True)
```



[ ]:

```
correct = 0
one_bit_error = 0
two_bit_error = 0
threemore_bit_error = 0
print('Shap of y_pred: ', te_Y.shape)
for i in range(len(pred_Y)):
    if pred_Y[i] == te_Y[i]:
        correct += 1
    elif abs(pred_Y[i] - te_Y[i]) == 1:
        one_bit_error += 1
    elif abs(pred_Y[i] - te_Y[i]) == 2:
        two_bit_error += 1
    else:
        threemore_bit_error += 1
plt.figure(figsize=(12, 12))
plt.bar(['correct', '1 bit error', '2 bit error', '3 bit error'], [correct,␣
 ↪one_bit_error, two_bit_error, threemore_bit_error])
plt.title('Prediction of test data')
plt.xlabel('Prediction')
plt.ylabel('Number of data')
plt.show()
```

```
Shap of y_pred:  (320,)
```


Prediction of test data
Number of data
correct
1 bit error
2 bit error
3 bit error
Prediction