Using Machine Learning for Finding out the Red Wine Quality

We all know about the significance of red wine these days. It is also better to know better about its quality and the factors which can affect its quality. In these terms, we can use our computer knowledge of ML, and predict ourselves the quality based on the different physicochemical properties. Now, this will be an estimation on the quality and people will be aware about that.

Personally, I feel like people should learn the concepts of ML not for something as hesitant, but for the study and learning the nature of relationships and predictions based on our algorithms. That way they can be confident in their results and seek better results not just on some particular aspects but on everything that has target and response variables.

Here, for the quality of red wine, I am using the dataset from 2009 from the Kaggle. I will be highlighting the procedure on the data visualization, preprocessing and model prediction and finally will focus on its significance.

I. Dataset

The dataset is from the following link in Kaggle: https://www.kaggle.com/datasets/uciml/red-wine-quality-cortez-et-al-2009?datasetId=4458&sortBy=voteCount

Here is a quick snapshot of the data frame.

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
...	...	...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	6
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

1599 rows × 12 columns

We carry a basis test on the data frame to check whether there are any null entries or not and are there any duplicated items and perform the summary statistics on the dataset and the result is as follows. There were no null entries but 240 duplicated rows and we dropped those duplicated rows.

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

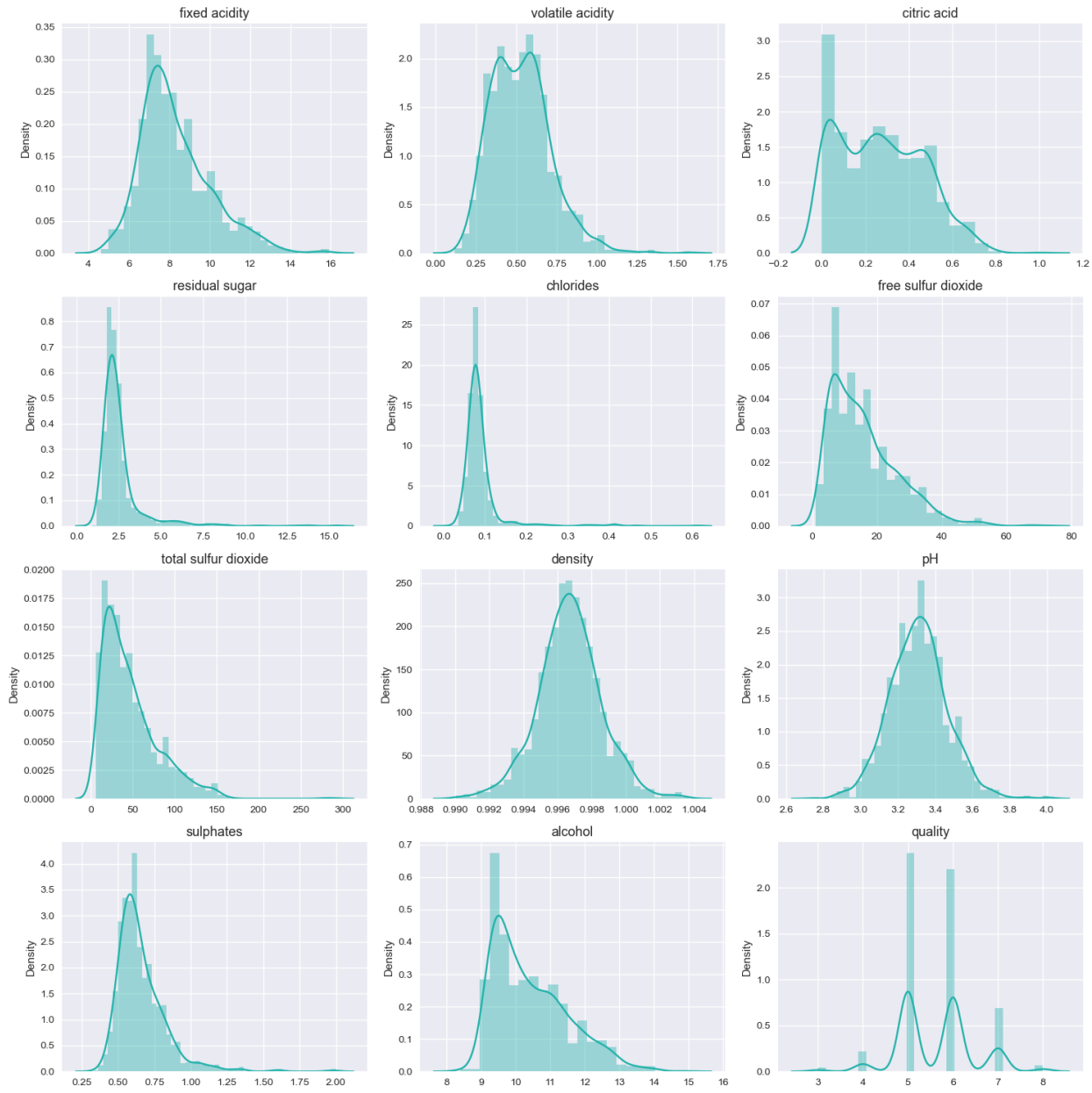
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   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
```

II. Data Preprocessing

Now, we import the libraries, and we did a bit of analysis on the data by visualizing the probability density as it gives us insight on the distribution of data. We have already imported pandas and displayed the data, so we won't be importing the dataset and named the dataset as redwine_df_cp.

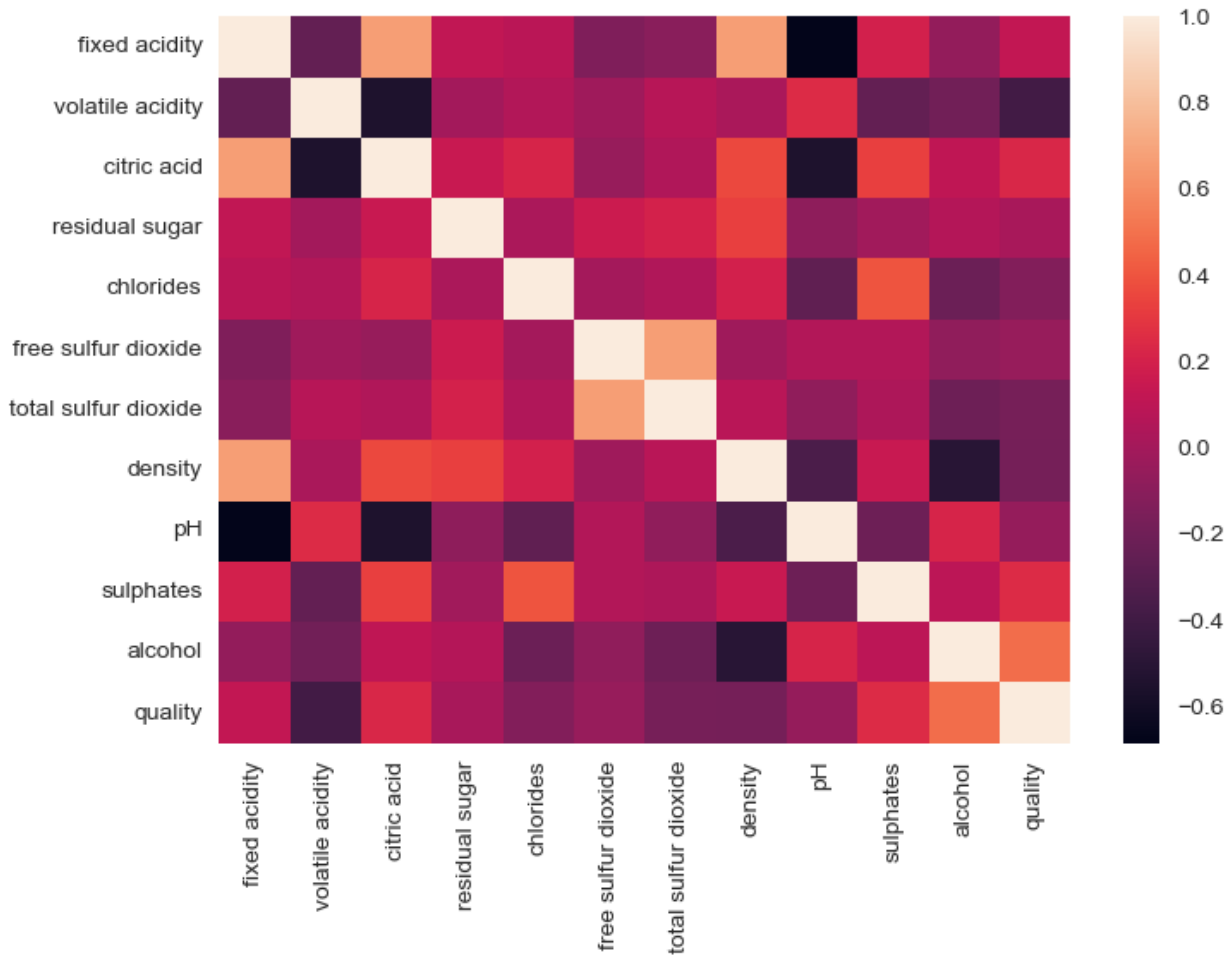
```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import warnings

warnings.filterwarnings('ignore')
plt.style.use('seaborn')
fig = plt.figure(figsize=(15,15))

for index, column in enumerate(redwine_df_cp.columns):
    plt.subplot(4,3,index+1)
    sns.distplot(x = redwine_df_cp.loc[:, column], color = 'lightseagreen')
    plt.title(column, size = 13)
    fig.tight_layout()
    plt.grid(True)
plt.show()
```

The result was as follows:



Here, we can see that there are some datasets like chlorides which are right tailed in distribution that might be due to outliers. Now for simplicity we don't touch outliers yet and will simply explore the dataset.

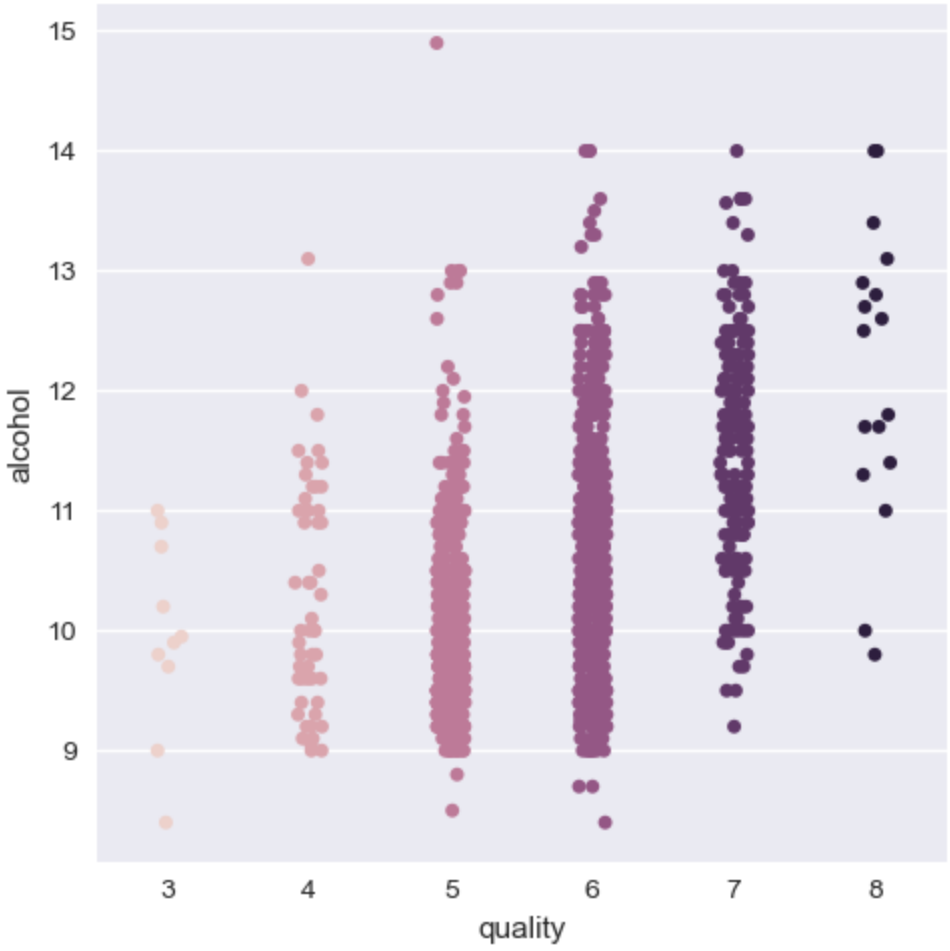
Now, we will try to check the correlation between different variables using the correlation coefficient and seaborn heatmap.

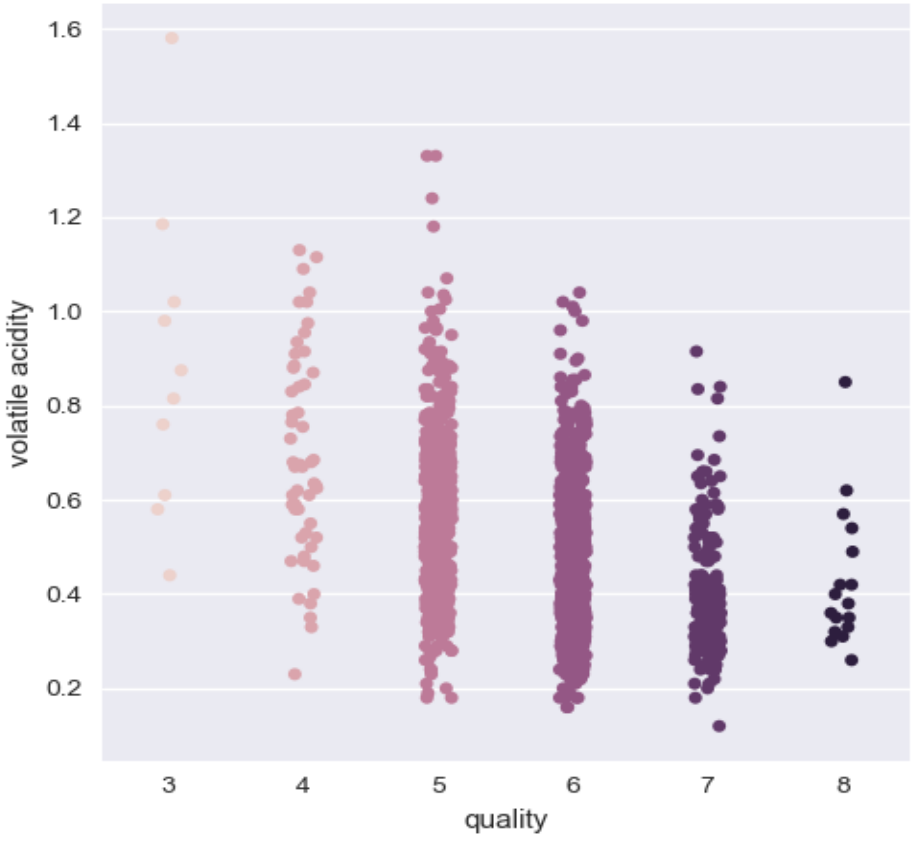


From the above heatmap, we can see that there is a positive correlation with alcohol of quality and negative with volatile acidity the most. So, we will be taking these two as prime factors for now and will be seeing their scatterplot/catplot and will see whether they are related or not.

```
sns.catplot(data=redwine_df_cp, x="quality", y="alcohol", hue="quality")
plt.show()
sns.catplot(data=redwine_df_cp, x="quality", y="volatile acidity", hue="quality")
plt.show()
```

The result of the categorical plot is as follows:


15
14
13
12
11
10
9
alcohol
3
4
5
6
7
8
quality


1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
volatile acidity
3
4
5
6
7
8
quality

Here we can see that there is indeed a positive correlation with quality of alcohol as we can think about and negative with volatile acidity. But it is better if we group the quality as a group of three categories as low, medium and high quality wines. That way data visualization and training is easy and less mundane.

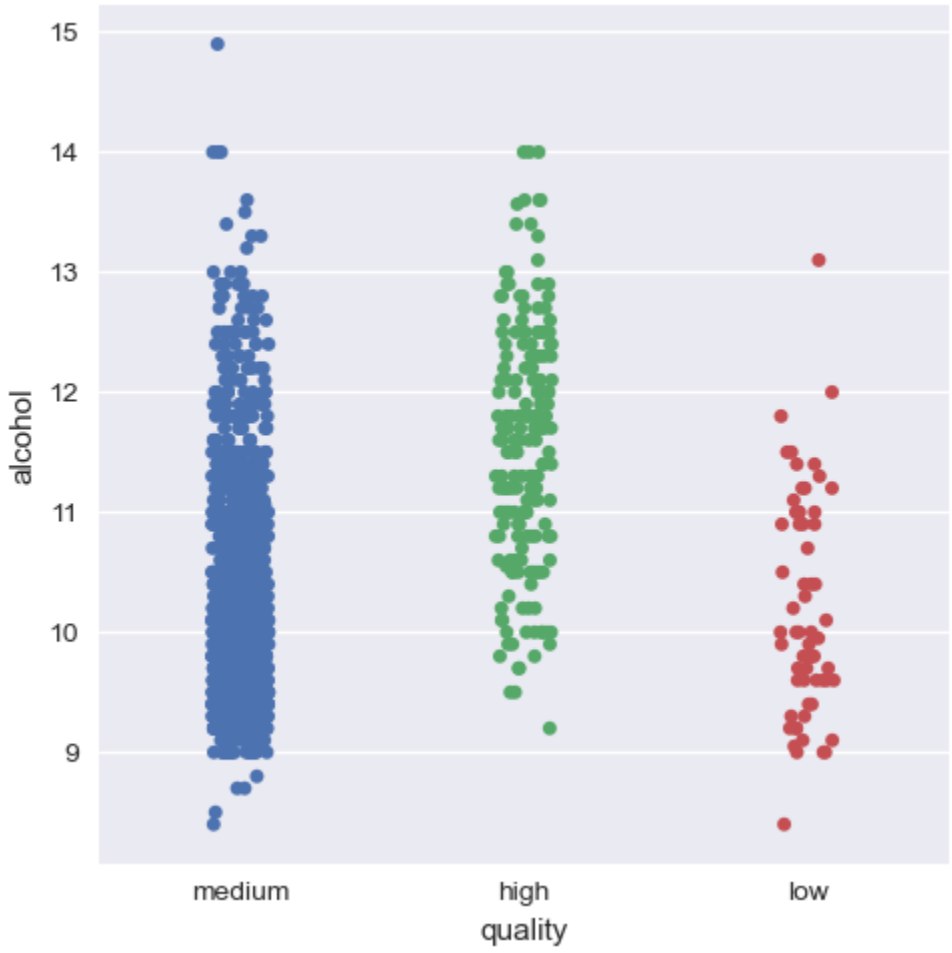
So, we create another copy of the dataframe to preserve the structure of the dataset.

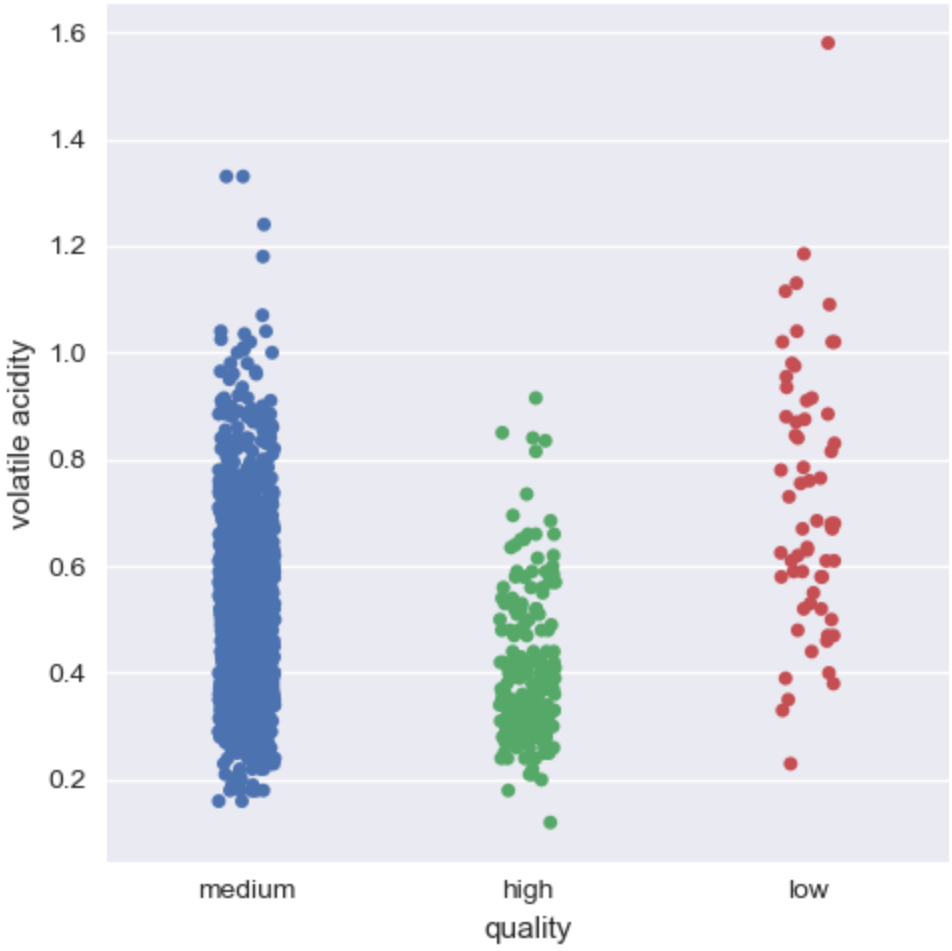
```
redwine_test_df = redwine_df_cp.copy()

redwine_test_df.head()
```

```
# Let's define a function to help us.

def quality_value(value):

    value = str(value)

    if value == '3':

        value = value.replace('3','low')

        return value

    elif value == '4':

        value = value.replace('4','low')

        return value

    elif value == '5':

        value = value.replace('5','medium')

        return value

    elif value == '6':

        value = value.replace('6','medium')

        return value

    elif value == '7':

        value = value.replace('7','high')

        return value
```

```
    elif value == '8':
        value = value.replace('8','high')
        return value

redwine_test_df['quality'] = redwine_test_df['quality'].apply(quality_value)
redwine_test_df["quality"].unique()
```

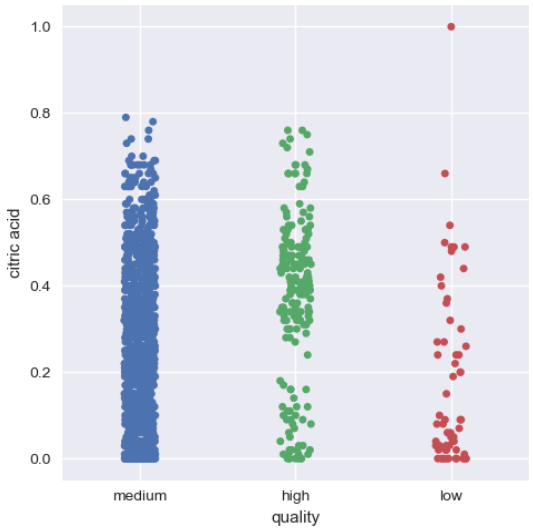
Now, we will again see the categorical plot of the alcohol and volatile acidity with quality.

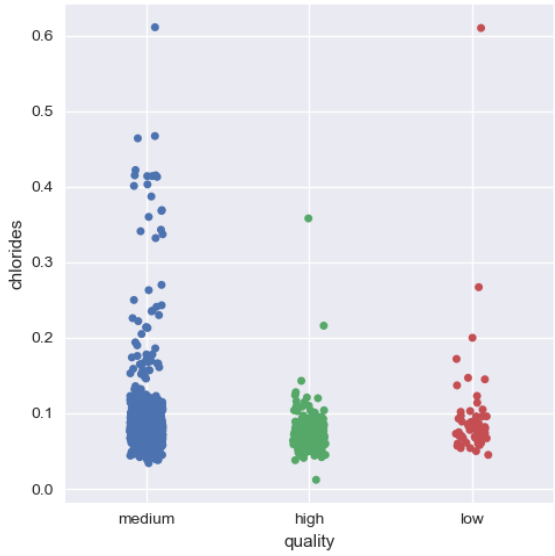


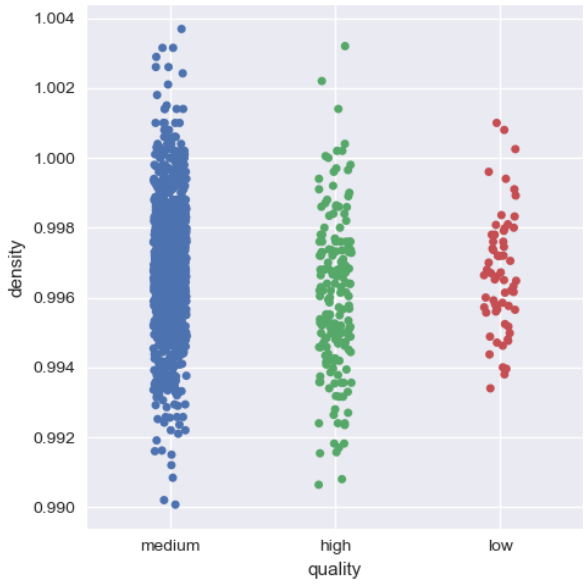


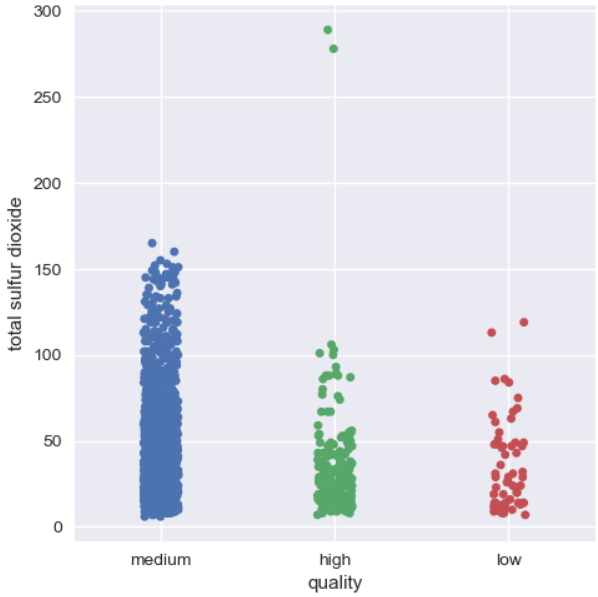
As we can see now, it's much more visual and definitive to say that there is indeed positive correlation and negative correlation of alcohol and volatile acidity with quality respectively.

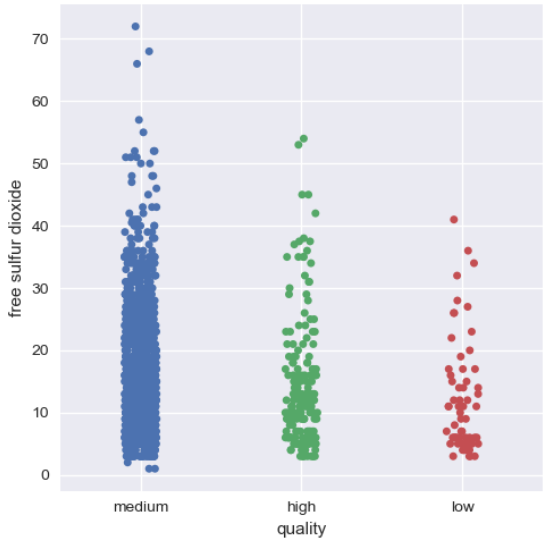
Now, we will visualize this for the rest of the variables.

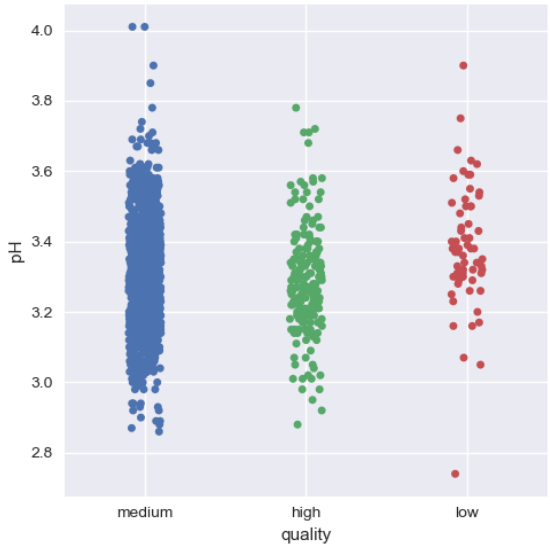










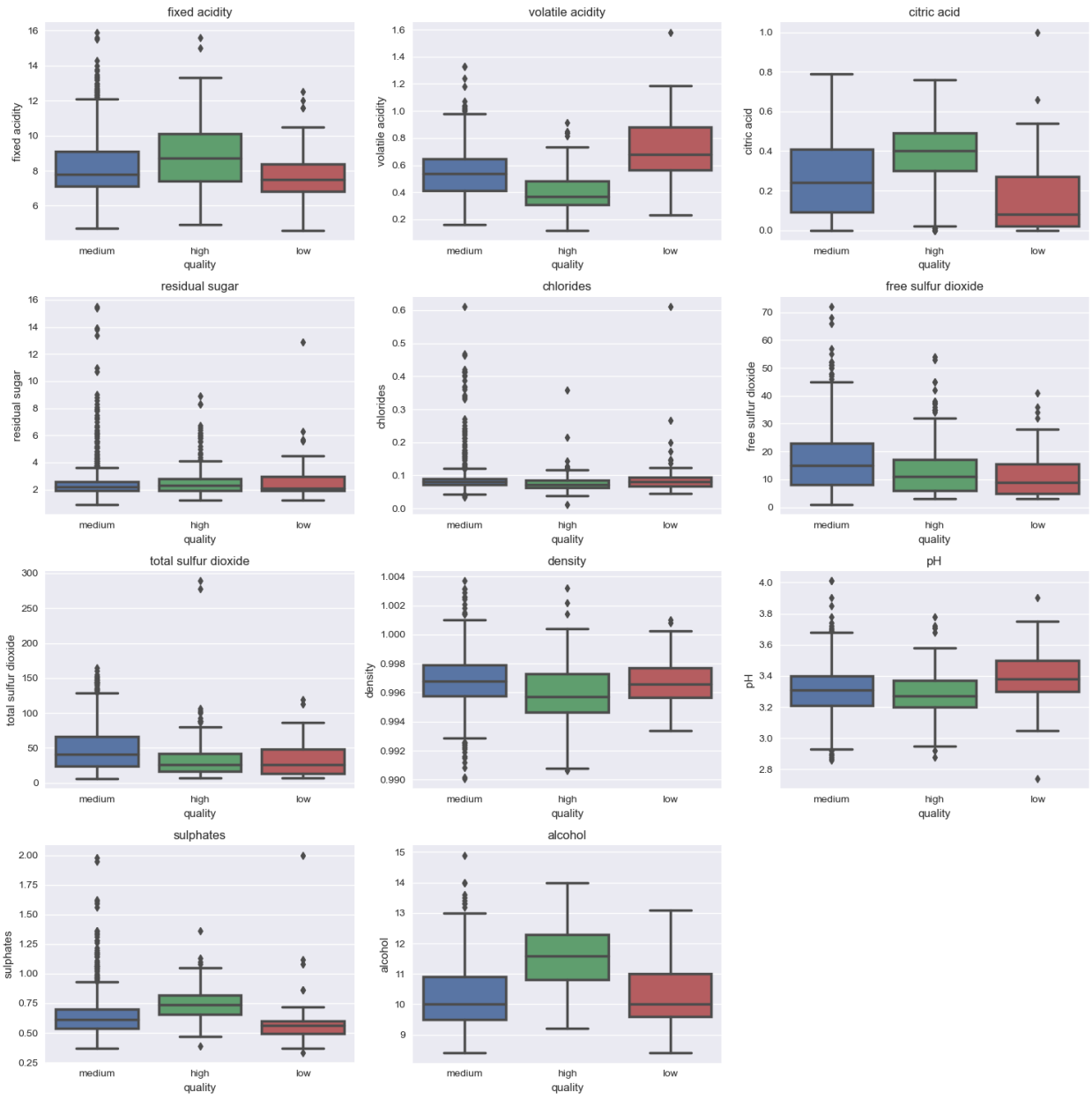


Hence, we can infer that as a whole all these factors contribute to the quality of red wine,with alcohol and volatile acidity as key factors.

Now, we will get onto seeing the dataset and visualizing the outliers for this we use seaborn boxplot.

```
fig = plt.figure(figsize=(15,15))

for index,column in enumerate(list(redwine_test_df.columns[:-1])):

    plt.subplot(4,3,index+1)
```

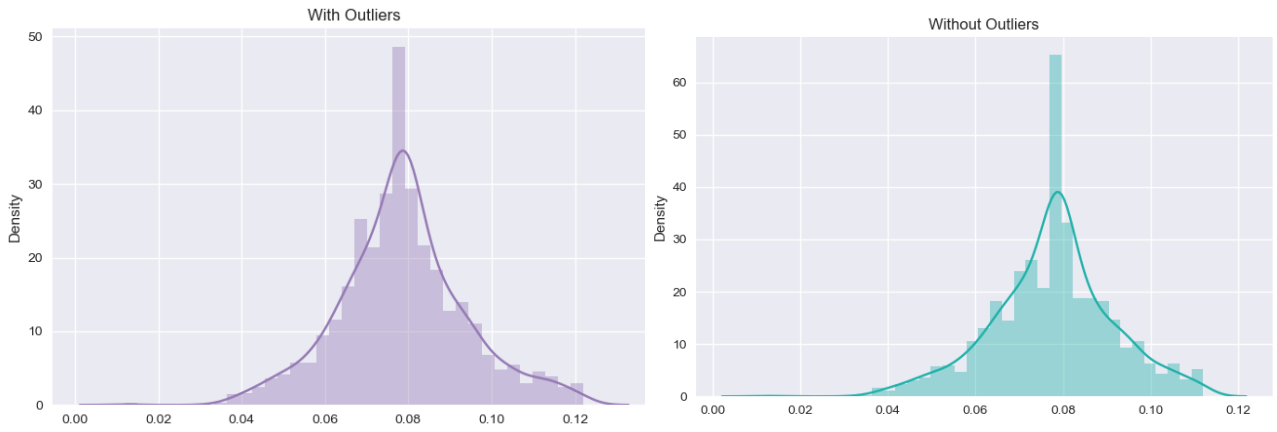
```
    sns.boxplot(y =redwine_test_df.loc[:, column], x =redwine_test_df['quality'],
linewidth=2.5)

    plt.title(column, size = 12)

fig.tight_layout()
```



Here, we can see the outliers in the physicochemical properties, but we won't clear them all out. We highlight the outlier with high distribution i.e. chlorides for this purpose and we will impute median in the outlier. The code is as follows:

```
df_chlorides = redwine_test_df['chlorides']

# distribution of the variable before cleaning

plt.figure(figsize=(8,5))

plt.title('With Outliers')

sns.distplot(x = df_chlorides, color = '#967bb6')

# setting the threshold value

Q1 = df_chlorides.quantile(0.25)

Q3 = df_chlorides.quantile(0.75)

# interquartile range

IQR = Q3 - Q1

lower_limit = Q1 - 1.5 * IQR

upper_limit = Q3 + 1.5 * IQR

# catching outliers

upper_outlier = (df_chlorides > upper_limit)

# determining the mean and median values

mean = df_chlorides.mean()

median = df_chlorides.median()

# assigning outliers to median due to its suitability

df_chlorides[upper_outlier] = median

# distribution of the data without outliers

plt.figure(figsize=(8,5))

plt.title('Without Outliers')

sns.distplot(x = df_chlorides, color = 'lightseagreen')
```

The result before and after is as follows for chloride column:



As, we can see the change in the dataset before and after the outlier. We will see its effect in the model performance and accuracy later on. For now, we will stick with other outliers as looking at the boxplot, other outliers don't have much impact on the model accuracy and performance.

III. Classification

Now, we import all the libraries required for classification.

```
from imblearn.over_sampling import SMOTE

from sklearn.preprocessing import StandardScaler

from sklearn.model_selection import train_test_split, cross_val_score

from sklearn.metrics import accuracy_score, confusion_matrix,

classification_report

import catboost as cb

import xgboost as xgb

import lightgbm as lgb

from sklearn.naive_bayes import GaussianNB

from sklearn.svm import SVC
```

```
from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.linear_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
```

Here some classifiers needs the quality column to be values such as 0, 1,2 so we switch it back accordingly.

```
redwine_test_df['quality'] = redwine_test_df['quality'].map({'high': 2, 'medium':
1, 'low': 0})

redwine_test_df['quality'].unique()
```

Now, we separate the dataset into dependent and independent variables.

```
y = redwine_test_df["quality"]

x = redwine_test_df.drop(["quality"], axis=1)

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=40)
```

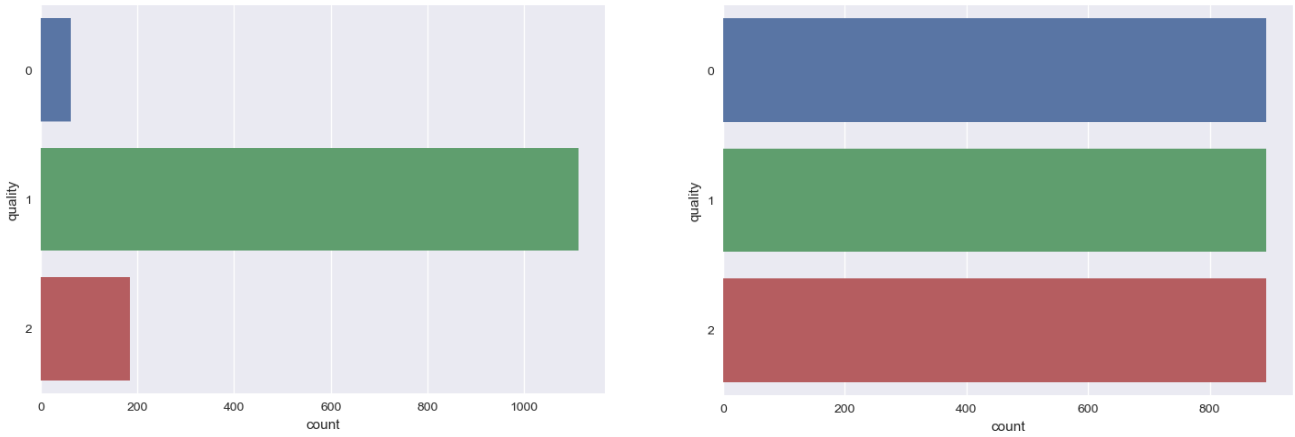
Now, the main thing here is Standardizing the data. The reason behind the standardization is making the data comparable by eliminating the difference in measurement units in the table. For this we use standard scalar.

```
scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)
```

Then, we use SMOTE technique for removing the class imbalance

```
smote = SMOTE(random_state=40)

X_train, y_train = smote.fit_resample(X_train, y_train)
```

```
sns.countplot(y=y_train)
plt.show()
```



Before and After SMOTE technique

Now, we use the models for applying the ML.

```
models = [
    ('Logistic Regression', LogisticRegression()),
    ('Support Vector Classifier', SVC()),
    ('Naive Bayes', GaussianNB()),
    ('KNN', KNeighborsClassifier()),
    ('Decision Tree', DecisionTreeClassifier()),
    ('Random Forest', RandomForestClassifier()),
    ('XGBoost', xgb.XGBClassifier()),
    ('LightGBM', lgb.LGBMClassifier()),
    ('CatBoost', cb.CatBoostClassifier(verbose=0)),
    ('AdaBoost', AdaBoostClassifier())
]
```

```
conf_matrices = []

class_reports = []

outcomes = []


for model_name, model in models:

    cv_results = cross_val_score(model, X_train, y_train, cv=5)

    mean_accuracy = cv_results.mean()


    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    accuracy = accuracy_score(y_test, y_pred)

    cm = confusion_matrix(y_test, y_pred)

    cr = classification_report(y_test, y_pred)


    conf_matrices.append((model_name, cm))

    class_reports.append((model_name, cr))

    outcomes.append((model_name, mean_accuracy, accuracy))


df_results = pd.DataFrame(outcomes, columns=['Model', 'Cross-Validation
Accuracy', 'Test Accuracy'])

df_results.sort_values('Cross-Validation Accuracy', ascending=False,
inplace=True)


df_results
```

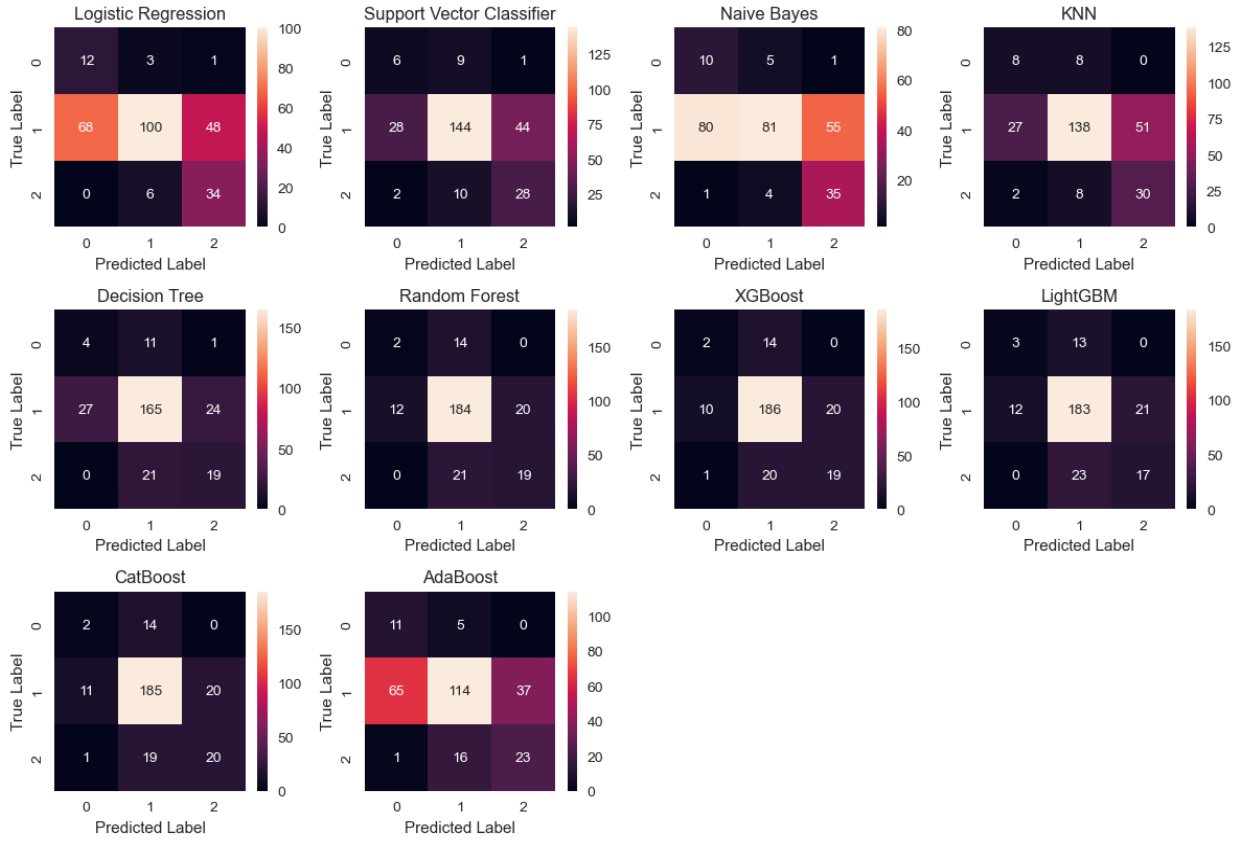
The result of the test accuracy and other reports are as follows:

	Model	Cross-Validation Accuracy	Test Accuracy
6	XGBoost	0.944950	0.761029
7	LightGBM	0.944207	0.746324
8	CatBoost	0.941600	0.761029
5	Random Forest	0.939361	0.753676
3	KNN	0.864579	0.647059
1	Support Vector Classifier	0.862368	0.654412
4	Decision Tree	0.858276	0.691176
0	Logistic Regression	0.708343	0.536765
9	AdaBoost	0.692703	0.544118
2	Naive Bayes	0.661463	0.463235

Here, we can see that CatBoost and XGBoost are high in test accuracy, but let's see the classification report and confusion matrices to find out which model to use and tune.

```
plt.figure(figsize=(12, 8))

for i, (model_name, cm) in enumerate(conf_matrices, 1):

    plt.subplot(3, 4, i)

    plt.title(model_name)

    sns.heatmap(cm, annot=True, fmt='d')

    plt.xlabel('Predicted Label')

    plt.ylabel('True Label')

plt.tight_layout()

plt.show()
```

The confusion matrices are depicted below:



The classification reports are as follows:

```
for model_name, cr in class_reports:

    print(f'\033[1m{model_name}:\033[0m')

    print(f'\033[1mClassification Report:\033[0m')

    print(cr)

    print('-'*60)
```

CatBoost:
Classification Report:

	precision	recall	f1-score	support
0	0.14	0.12	0.13	16
1	0.85	0.86	0.85	216
2	0.50	0.50	0.50	40
accuracy			0.76	272
macro avg	0.50	0.49	0.50	272
weighted avg	0.76	0.76	0.76	272

AdaBoost:
Classification Report:

	precision	recall	f1-score	support
0	0.14	0.69	0.24	16
1	0.84	0.53	0.65	216
2	0.38	0.57	0.46	40
accuracy			0.54	272
macro avg	0.46	0.60	0.45	272
weighted avg	0.74	0.54	0.60	272

XGBoost:
Classification Report:

	precision	recall	f1-score	support
0	0.15	0.12	0.14	16
1	0.85	0.86	0.85	216
2	0.49	0.47	0.48	40
accuracy			0.76	272
macro avg	0.50	0.49	0.49	272
weighted avg	0.75	0.76	0.76	272

LightGBM:
Classification Report:

	precision	recall	f1-score	support
0	0.20	0.19	0.19	16
1	0.84	0.85	0.84	216
2	0.45	0.42	0.44	40
accuracy			0.75	272
macro avg	0.49	0.49	0.49	272
weighted avg	0.74	0.75	0.74	272

Naive Bayes:
Classification Report:

	precision	recall	f1-score	support
0	0.11	0.62	0.19	16
1	0.90	0.38	0.53	216
2	0.38	0.88	0.53	40
accuracy			0.46	272
macro avg	0.46	0.62	0.42	272
weighted avg	0.78	0.46	0.51	272

KNN:
Classification Report:

	precision	recall	f1-score	support
0	0.22	0.50	0.30	16
1	0.90	0.64	0.75	216
2	0.37	0.75	0.50	40
accuracy			0.65	272
macro avg	0.49	0.63	0.51	272
weighted avg	0.78	0.65	0.68	272

Decision Tree:
Classification Report:

	precision	recall	f1-score	support
0	0.13	0.25	0.17	16
1	0.84	0.76	0.80	216
2	0.43	0.47	0.45	40
accuracy			0.69	272
macro avg	0.47	0.50	0.47	272
weighted avg	0.74	0.69	0.71	272

Random Forest:
Classification Report:

	precision	recall	f1-score	support
0	0.14	0.12	0.13	16
1	0.84	0.85	0.85	216
2	0.49	0.47	0.48	40
accuracy			0.75	272
macro avg	0.49	0.48	0.49	272
weighted avg	0.75	0.75	0.75	272

Logistic Regression:
Classification Report:

	precision	recall	f1-score	support
0	0.15	0.75	0.25	16
1	0.92	0.46	0.62	216
2	0.41	0.85	0.55	40
accuracy			0.54	272
macro avg	0.49	0.69	0.47	272
weighted avg	0.80	0.54	0.58	272

Support Vector Classifier:
Classification Report:

	precision	recall	f1-score	support
0	0.17	0.38	0.23	16
1	0.88	0.67	0.76	216
2	0.38	0.70	0.50	40
accuracy			0.65	272
macro avg	0.48	0.58	0.50	272
weighted avg	0.77	0.65	0.69	272

Now, looking at the classification report, we chose CatBoost as our classifier and as our model. Now, we tune the model for its best performance and accuracy. For this we use optuna for tuning the model.

```
import optuna

def objective(trial):

    model = cb.CatBoostClassifier(

        iterations=trial.suggest_int("iterations", 100, 1000),

        learning_rate=trial.suggest_float("learning_rate", 1e-3, 1e-1, log=True),

        depth=trial.suggest_int("depth", 4, 10),

        l2_leaf_reg=trial.suggest_float("l2_leaf_reg", 1e-8, 100.0, log=True),

        bootstrap_type=trial.suggest_categorical("bootstrap_type", ["Bayesian"]),

        random_strength=trial.suggest_float("random_strength", 1e-8, 10.0,
log=True),

        bagging_temperature=trial.suggest_float("bagging_temperature", 0.0,
10.0),

        od_type=trial.suggest_categorical("od_type", ["IncToDec", "Iter"]),

        od_wait=trial.suggest_int("od_wait", 10, 50),

        verbose=False

    )

    model.fit(X_train, y_train)

    y_pred = model.predict(X_test)

    return accuracy_score(y_test, y_pred)

study = optuna.create_study(direction='maximize')

study.optimize(objective, n_trials=30)

print('Best hyperparameters:', study.best_params)

print('Best Accuracy:', study.best_value)
```

After this, the tuned parameters were passed into the model and we got the final accuracy of the model and our model was devised to the test accuracy of 77.57% which is higher.

```
model = cb.CatBoostClassifier(**study.best_params)

model.fit(X_train, y_train)

y_pred = model.predict(X_test)

acc_score = accuracy_score(y_test, y_pred)

acc_score
```

Hence, we can now predict the quality from the array of constituents value.

```
# taking the input

df_columns = list(redwine_test_df.columns)

test_array = []

for col_name in df_columns:

    if col_name == "quality":

        continue

    else:

        data = float(input(f"Enter the {col_name} quantity
[{redwine_test_df[col_name].min()} - {redwine_test_df[col_name].max()}]: "))

        test_array.append(data)


# giving the data to our model

q_value = model.predict(test_array)

if q_value == 1:

    print("The red wine is of medium quality!")

elif q_value == 2:
```

```
    print("The red wine is of high quality!")
else:
    print("The red wine is of poor quality!")
```

Here, we tested a random sample of wine and we got the wine quality as average. Clearly helping with the outliers of the chloride really made improvement in our model.

IV. Significance

So, the question may arise about its significance. Companies can use this technique of ML for creating better models and by using the dataset with thousands of data. This is just a simple dataset for devising a method for relating the quality of red wine with its physicochemical properties. General laymen can indeed know about what factors can affect the quality of red wine by analyzing the dataset in the visualization part. These algorithms and techniques can not only be used here but in every field of response and target variables, which can indeed boost our capabilities.