**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.

1. **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.A screenshot of a computer

Description automatically generated

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. A screen shot of a black and white screen

Description automatically generated

A screenshot of a computer

Description automatically generated

1. **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')

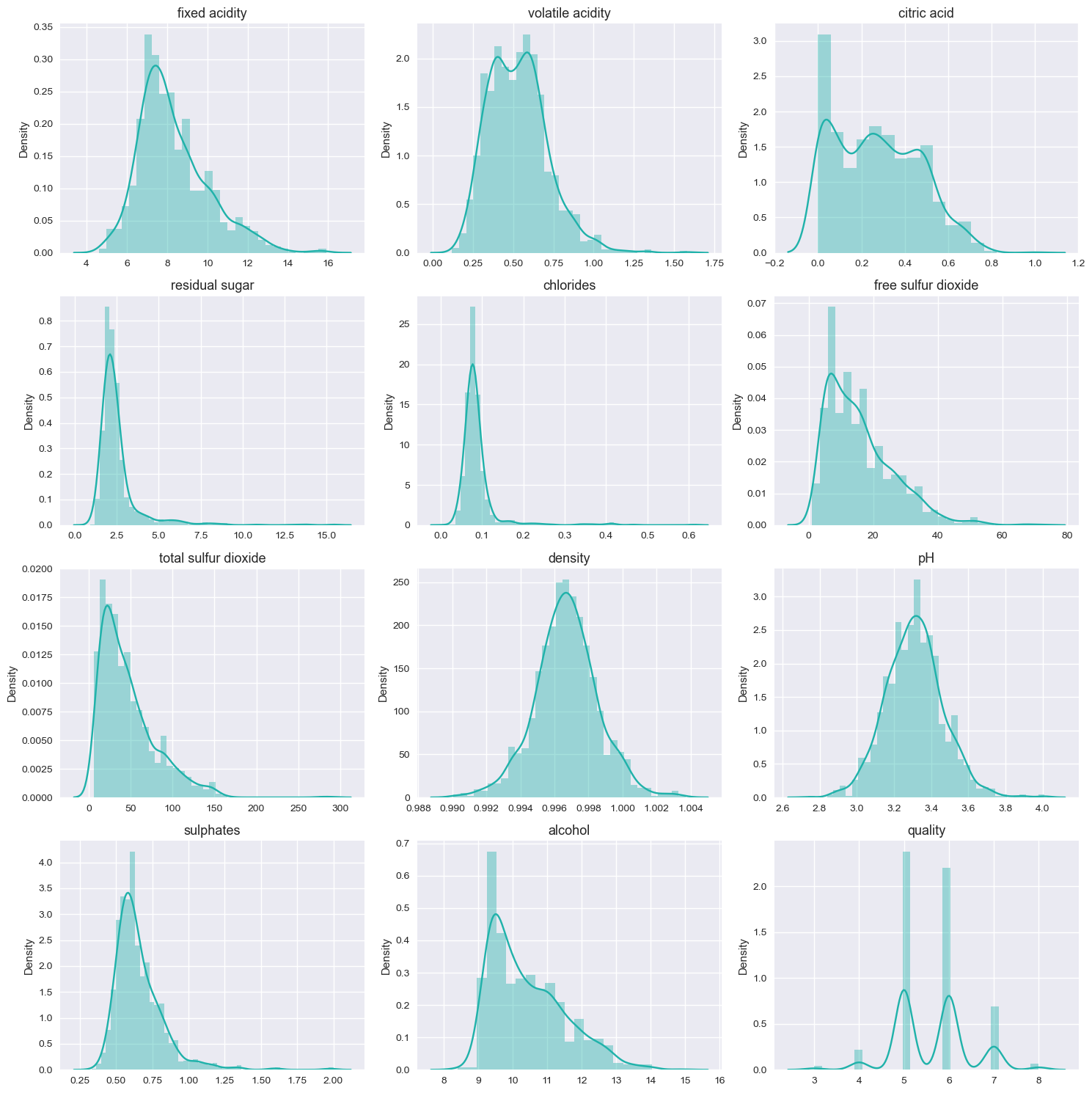
    plt.title(column, size = 13)

    fig.tight\_layout()

    plt.grid(True)

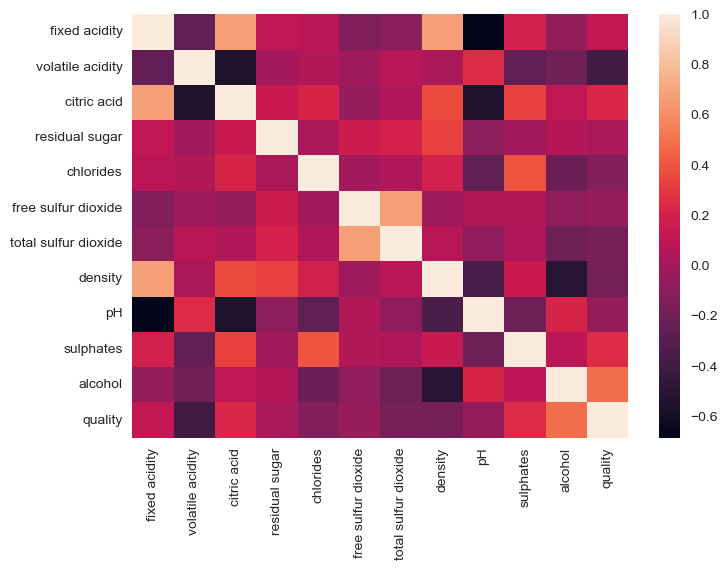
plt.show()

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.



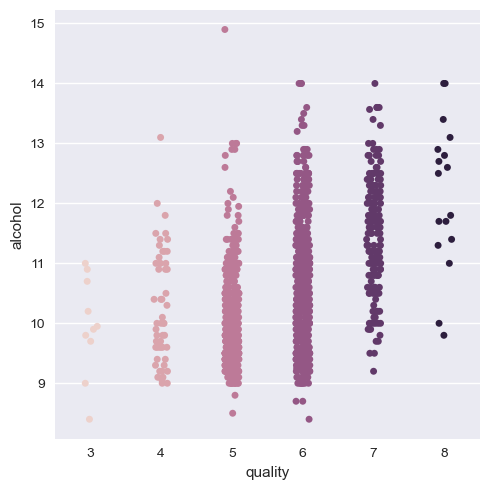
From the above heatmap, we can see that there is a positive correlation with alcohol of quality and negative with volatile acidity the most. So, we will be taking these two as prime factors for now and will be seeing their scatterplot/catplot and will see whether they are related or not.

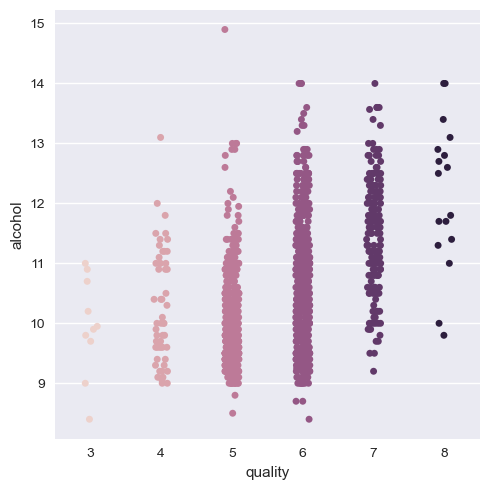
sns.catplot(data=redwine\_df\_cp, x="quality", y="alcohol", hue="quality")

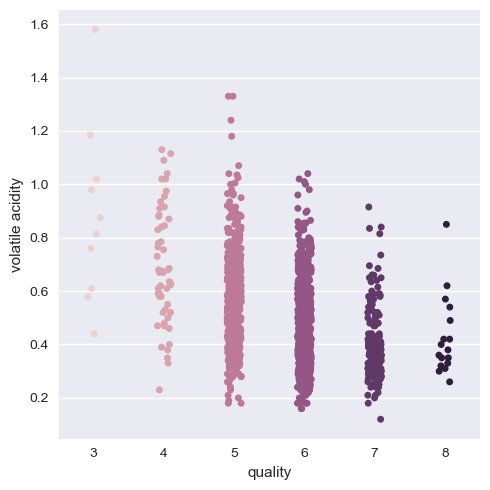
plt.show()

sns.catplot(data=redwine\_df\_cp, x="quality", y="volatile acidity", hue="quality")

plt.show()

The result of the categorical plot is as follows:





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

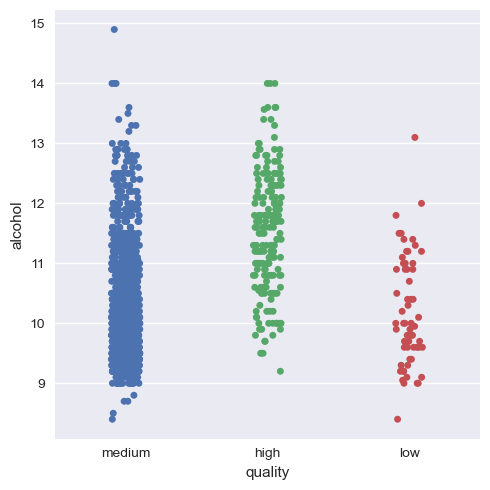
elif value == '8':

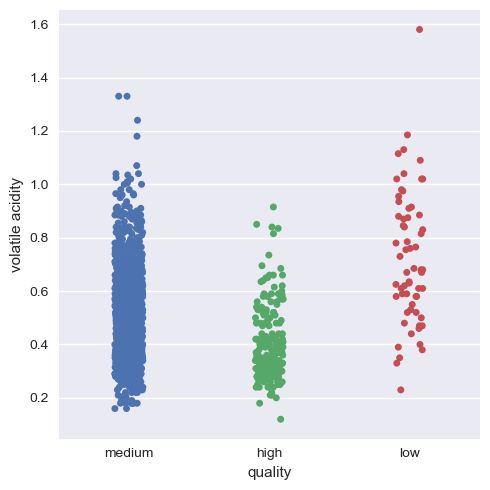
value = value.replace('8','high')

return value

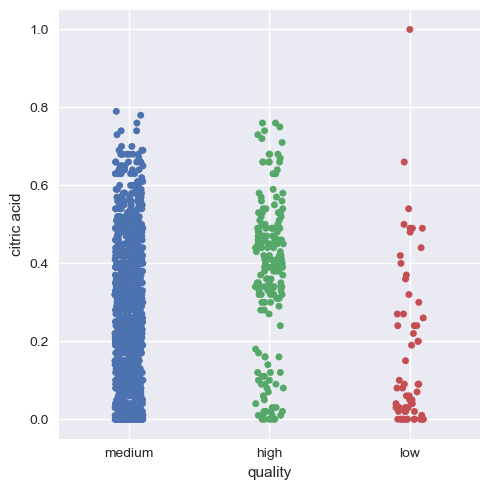
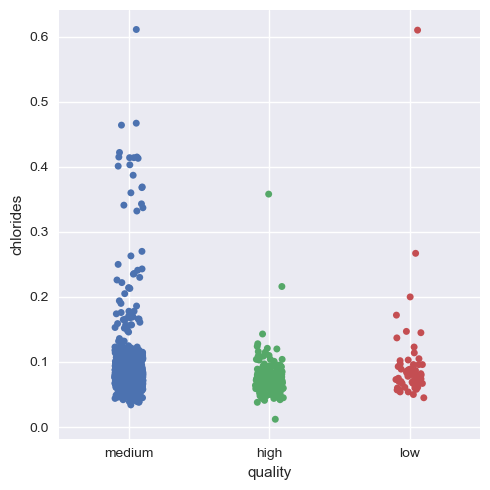
redwine\_test\_df['quality'] = redwine\_test\_df['quality'].apply(quality\_value)

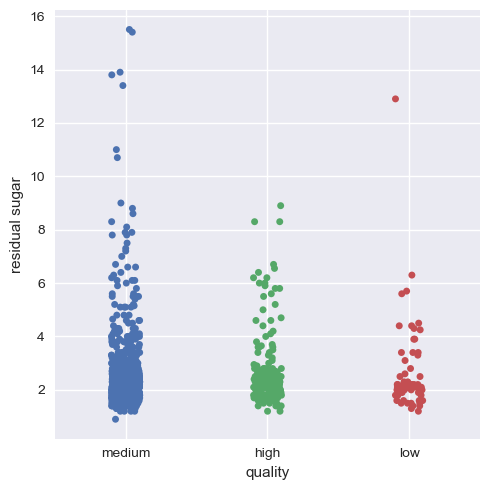
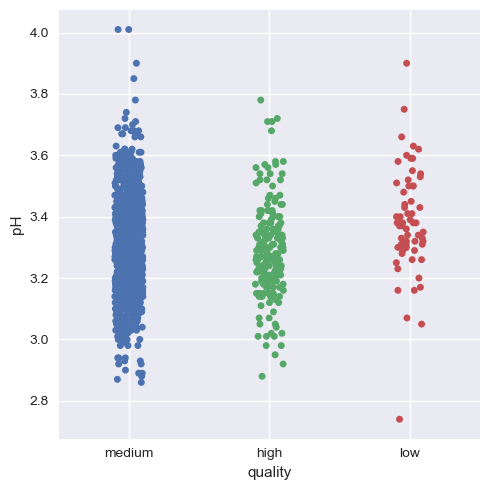
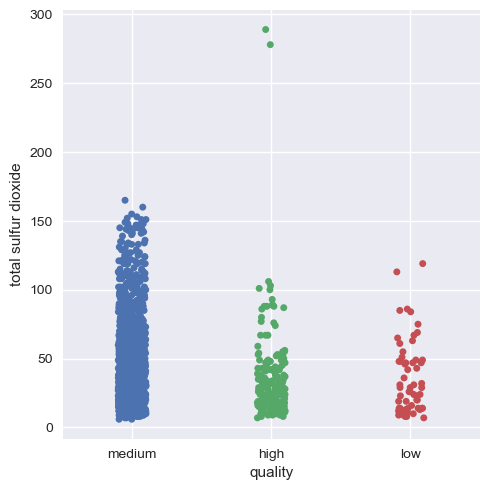
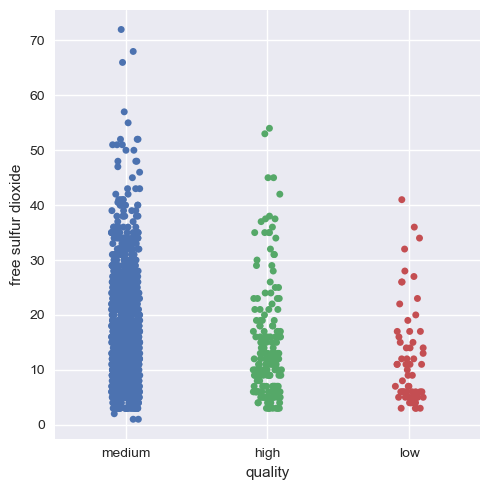
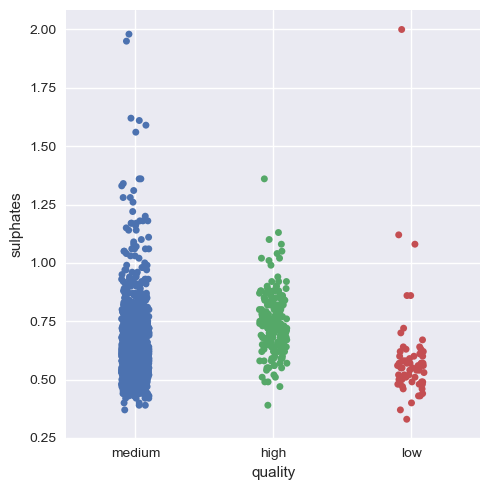
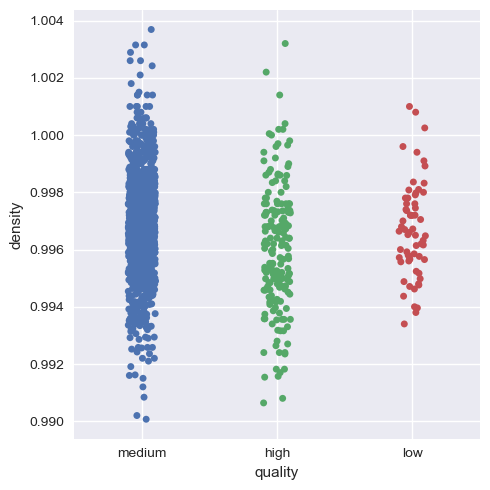
redwine\_test\_df["quality"].unique()

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))

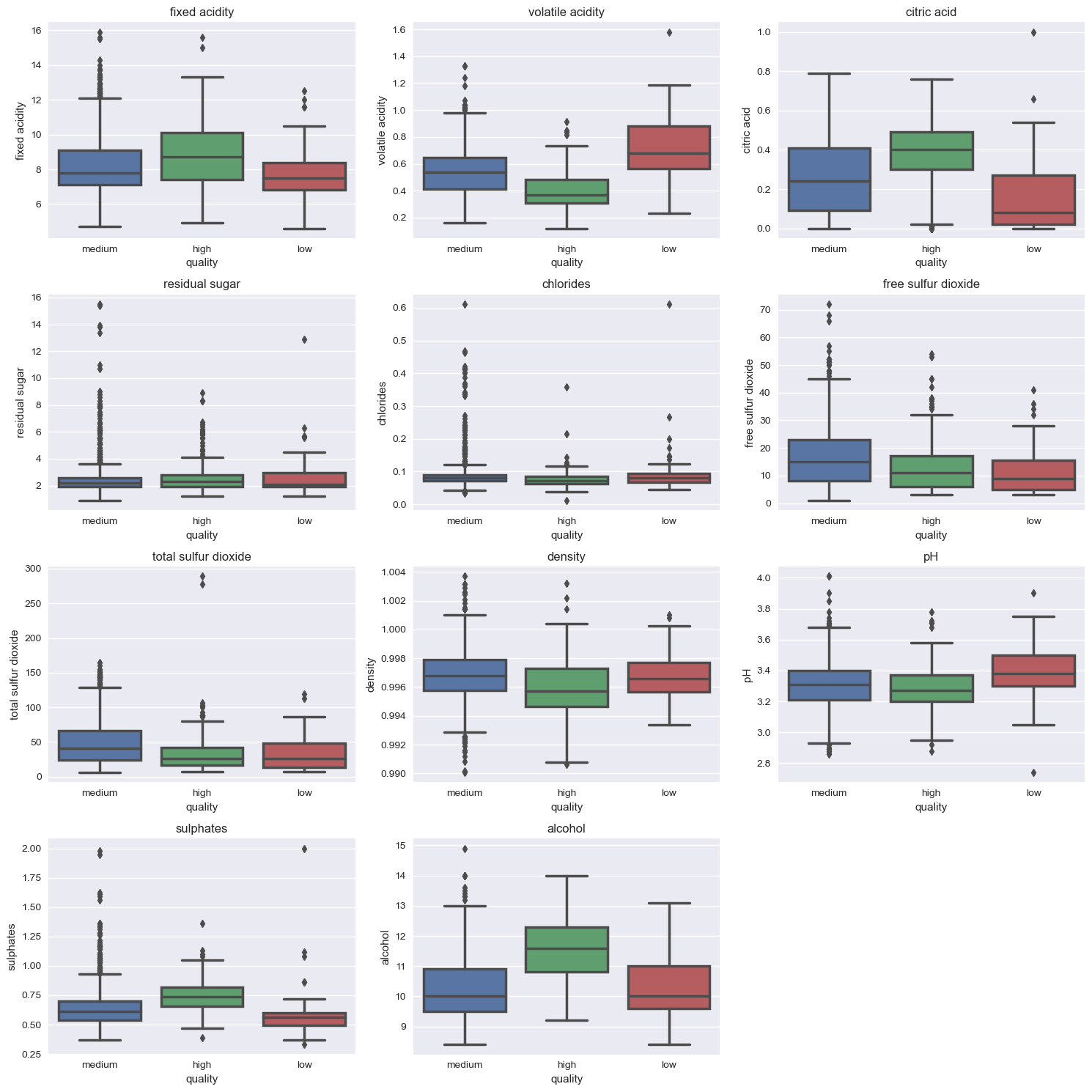
for index,column in enumerate(list(redwine\_test\_df.columns[:-1])):

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

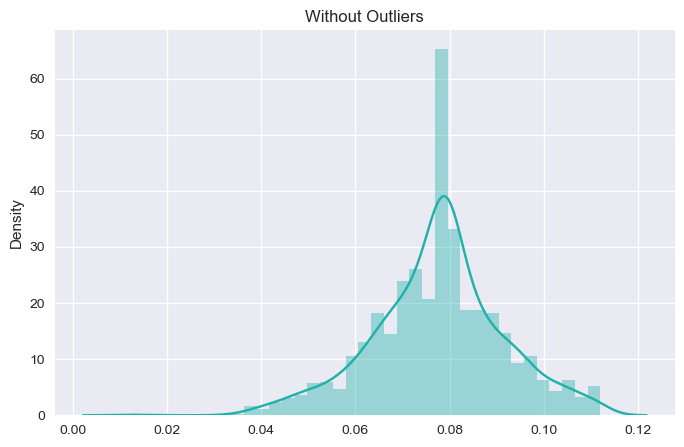
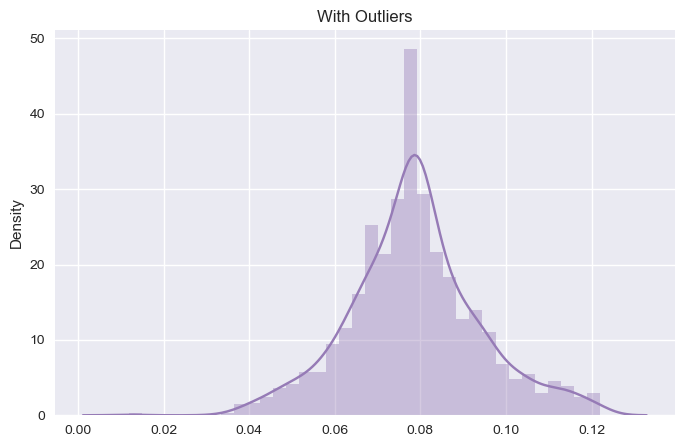
# distribution of the data without outliers

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)

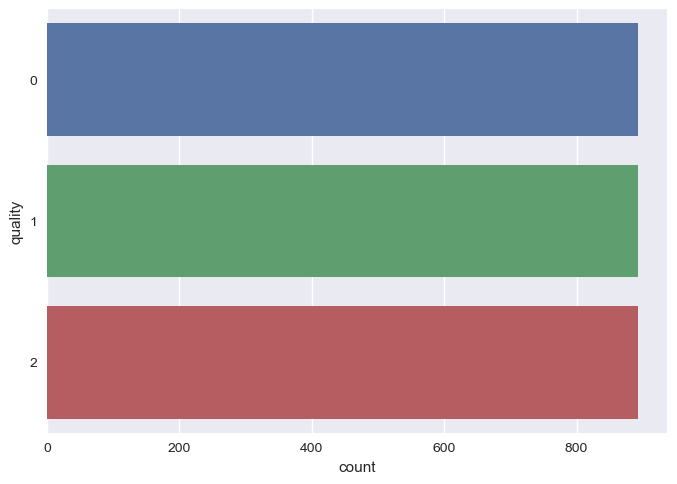
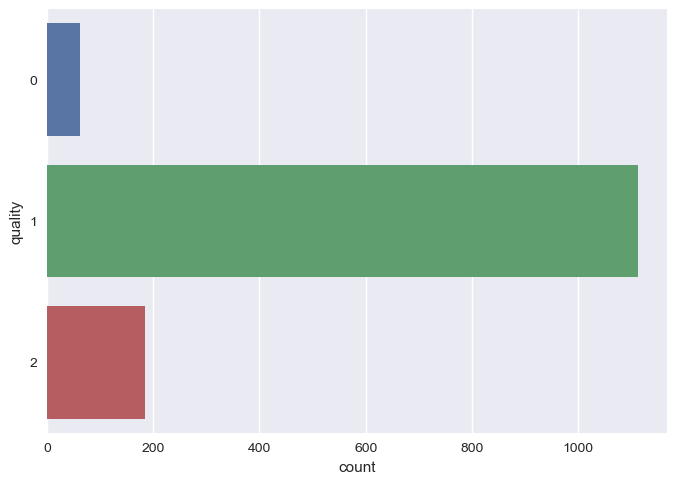
Then, we use SMOTE technique for removing the class imbalance

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))

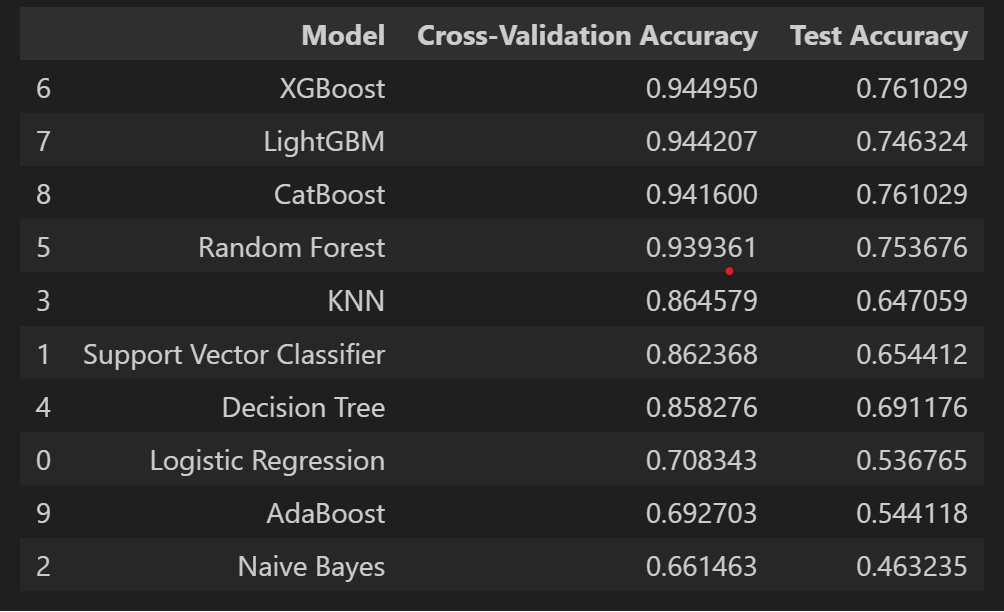
outcomes.append((model\_name, mean\_accuracy, accuracy))

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:

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')

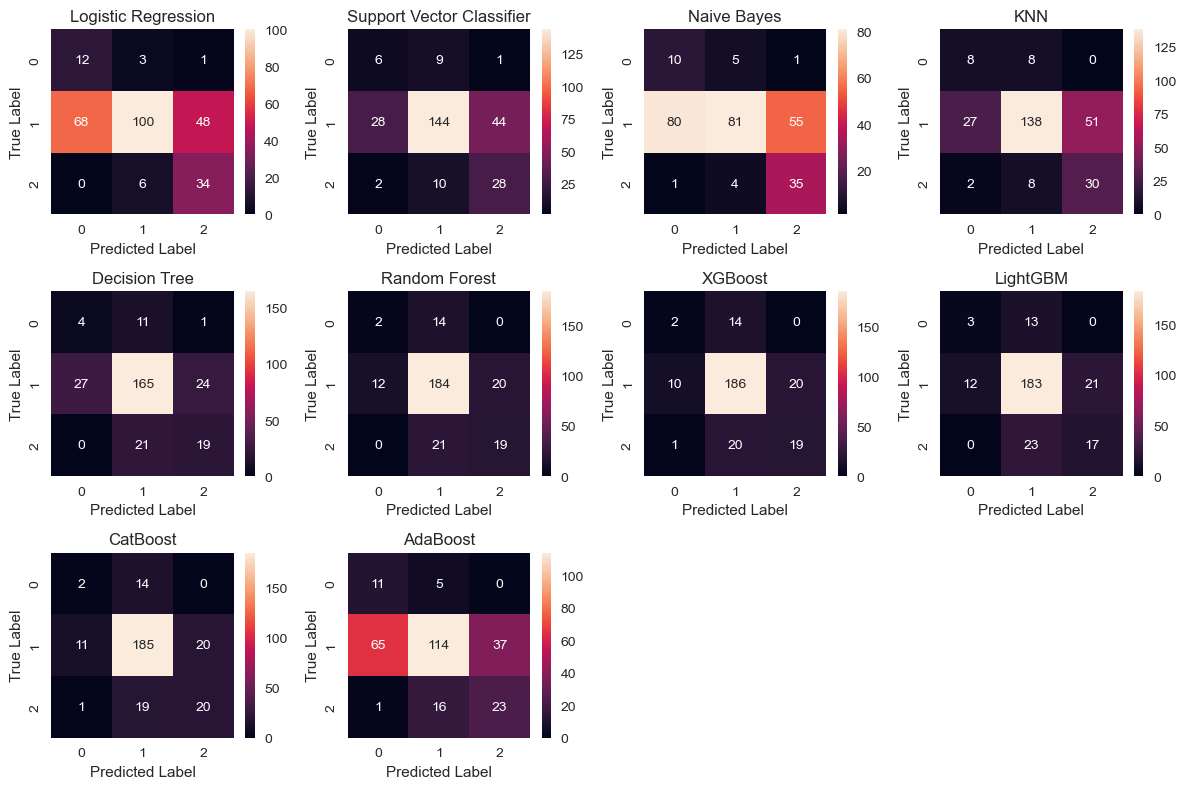
plt.xlabel('Predicted Label')

plt.ylabel('True Label')

plt.tight\_layout()

plt.show()

The confusion matrices are depicted below:



The classification reports are as follows:

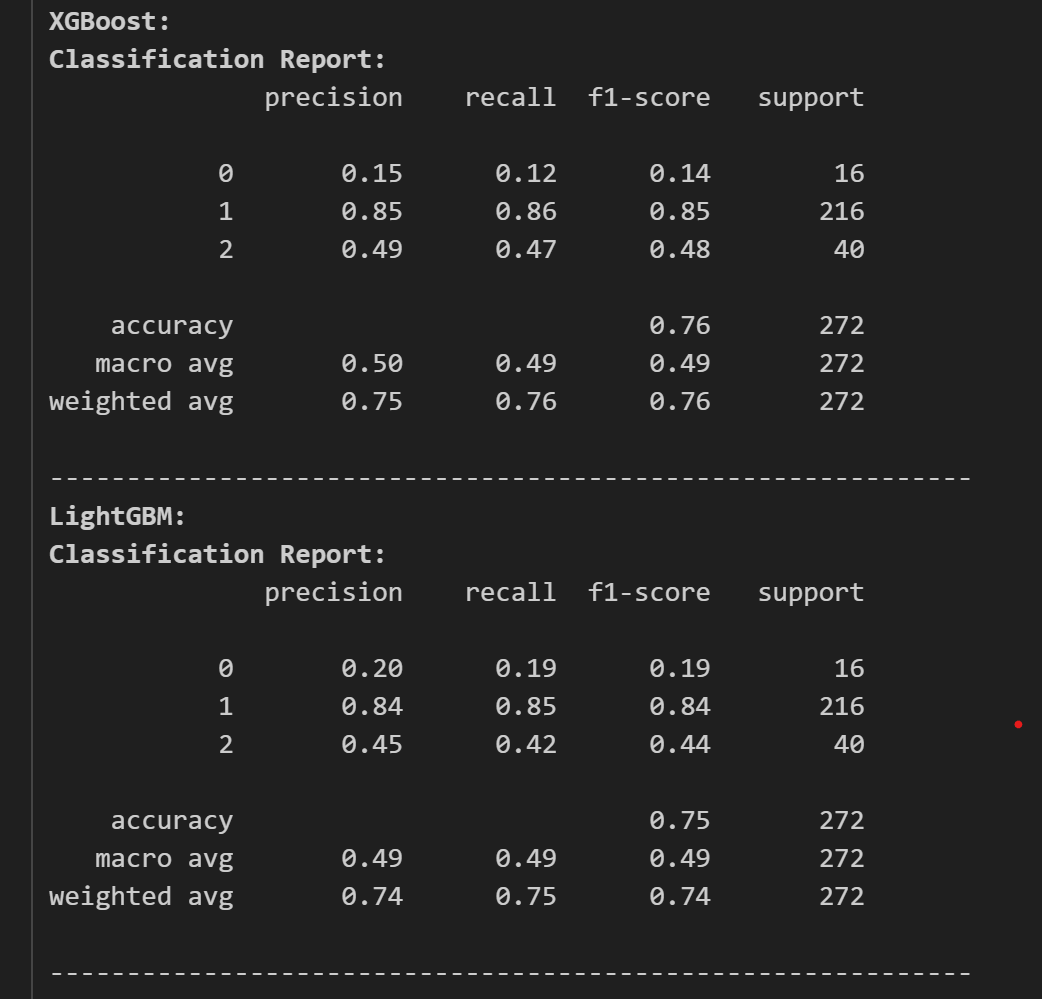
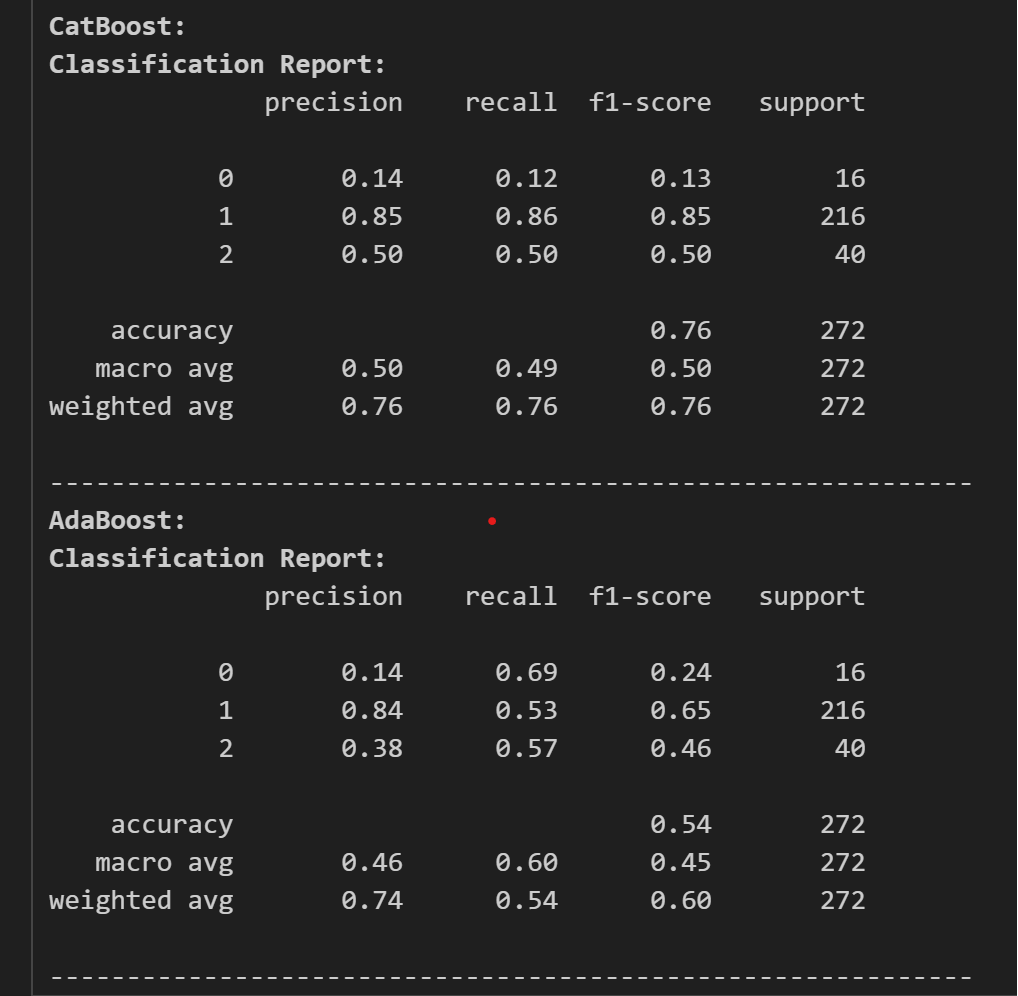
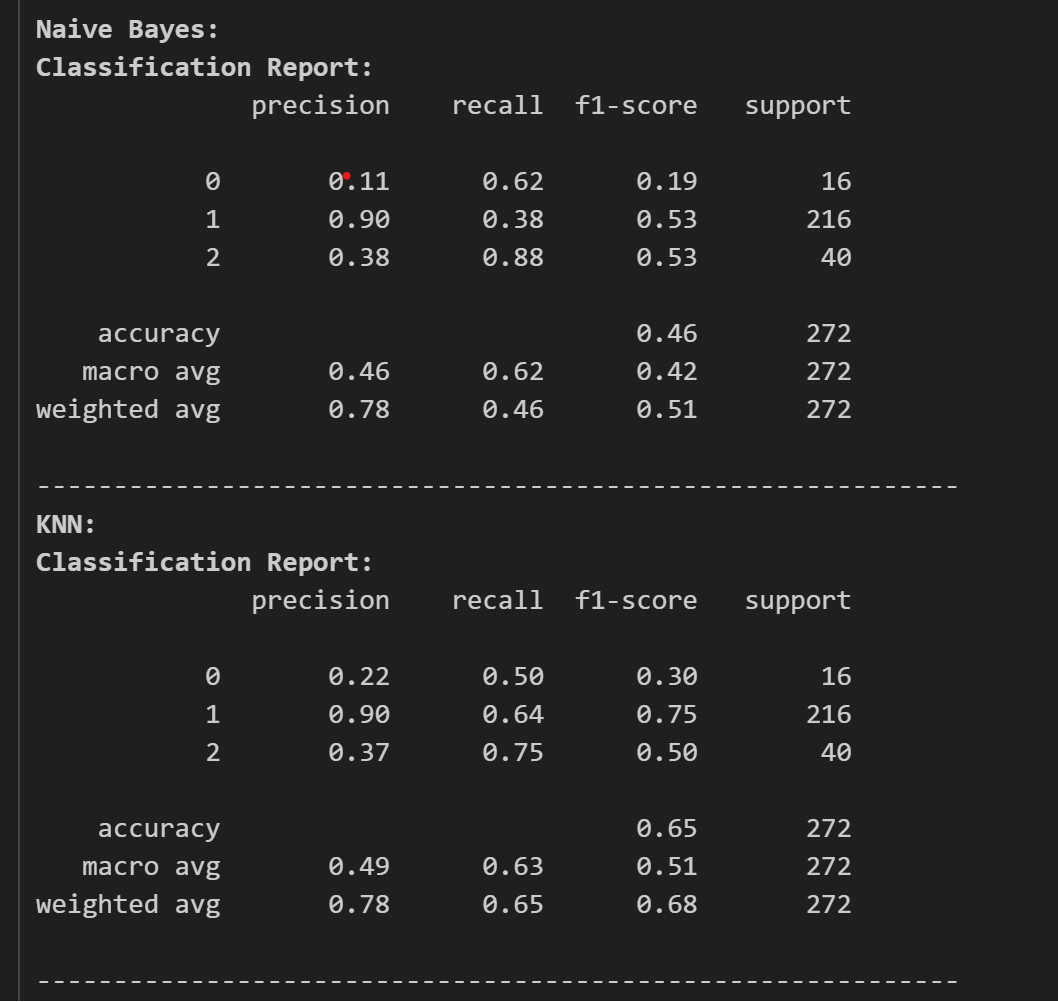
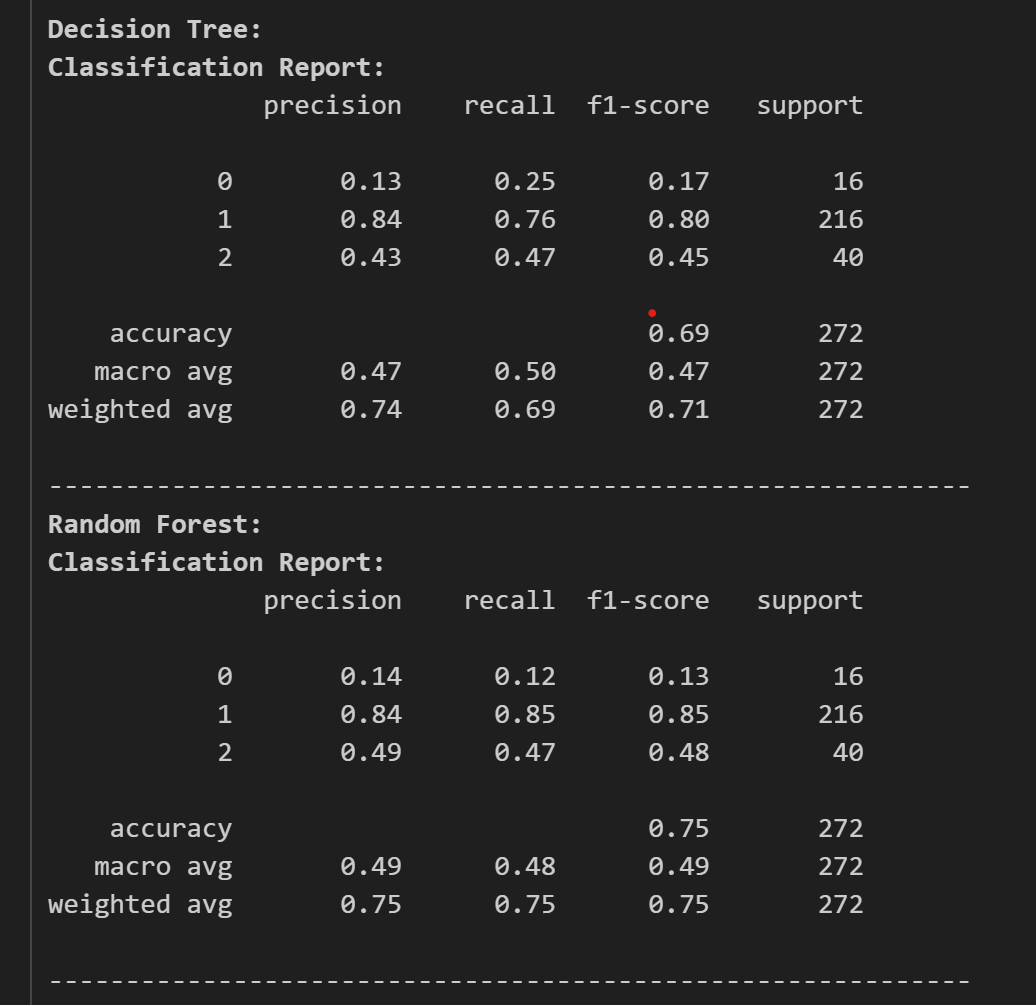
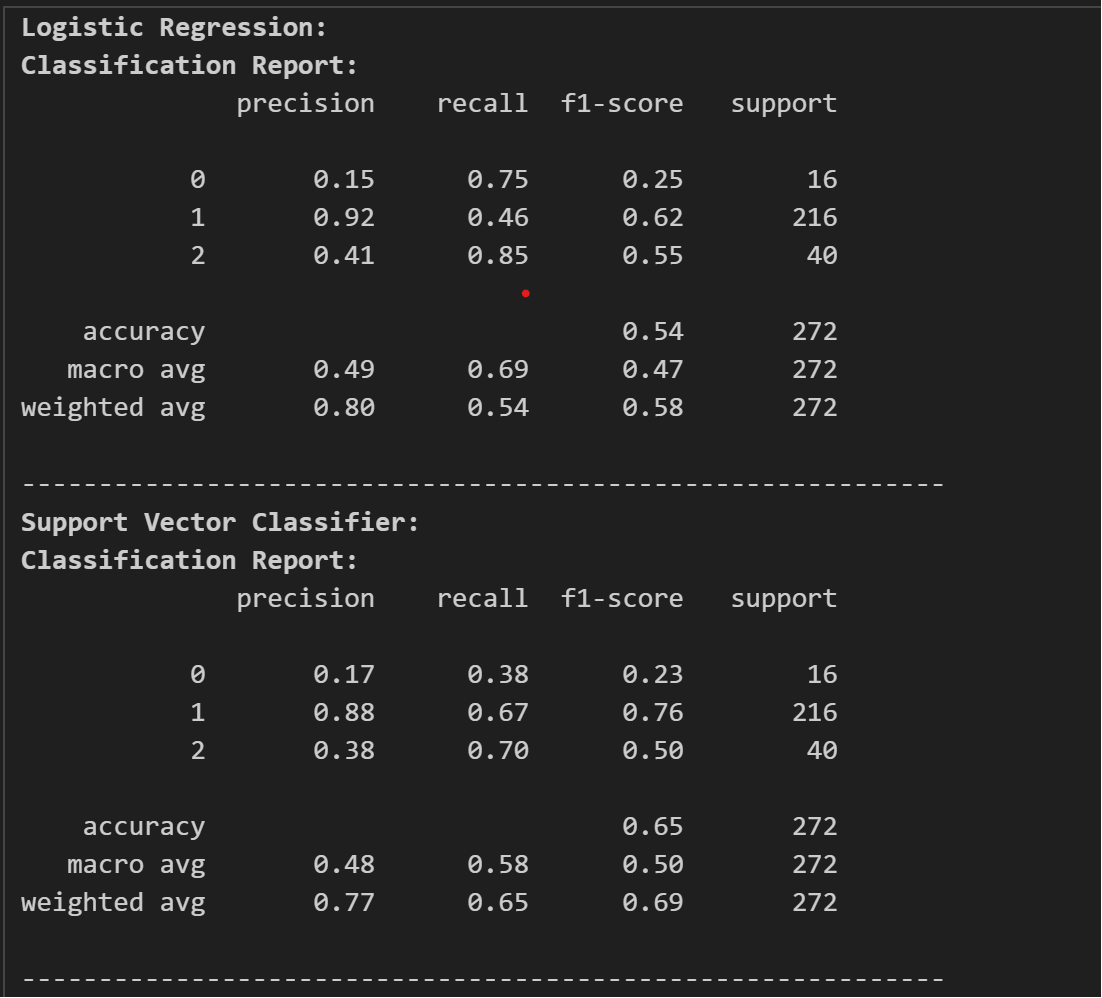
for model\_name, cr in class\_reports:

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

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

print(cr)

print('-'\*60)



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.