Raw Wine Quality Prediction Project

In [58]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.stats as stats
from scipy.stats import zscore

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.neighbors import KNeighborsClassifier


from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV

import warnings
warnings.filterwarnings("ignore")
import joblib
```

In [6]:
```
df=pd.read_csv("https://raw.githubusercontent.com/dsrscientist/DSData/master/winequality-red.csv"
```

In [7]:
```
df
```

Out[7]:

	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

In [8]:
```
df.shape
```

Out[8]:
```
(1599, 12)
```

There are total 1599 rows and 12 columns present in our dataset

In [9]:
```
df.isnull().sum()
```

Out[9]:
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

We do not see any missing values in any of the columns of our dataset so there is no need to handle missing data.

In [10]:
```
df.info()
```

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

In [11]:
```
df.describe()
```

Out[11]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.0(
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.3
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.1
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.7
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.2
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.3
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.4(
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.0

Using the describe method I can see the count, mean, standard deviation, minimum, maximum and inter quantile values of our dataset. observation:

1. There is a big gap between 75% and max values of residual sugar column.

2. There is a big gap between 75% and max values of free sulfur dioxide column.

3. There is a huge gap between 75% and max value of total sulfur dioxide column.

All these gaps indicates that there are outliers present in our dataset which might need to be treated to get better model accuracy.

In [12]:
```
df.skew()
```

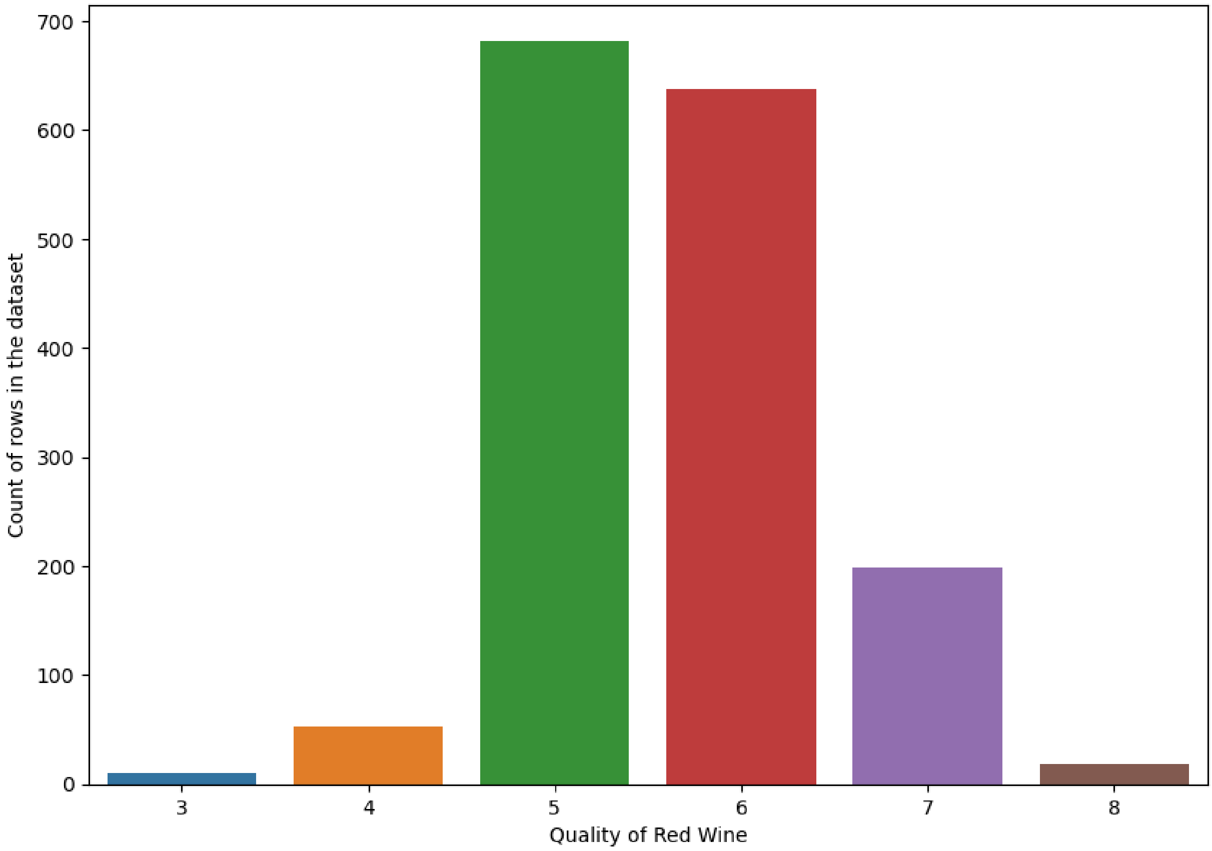
Out[12]:
```
fixed acidity           0.982751
volatile acidity        0.671593
citric acid             0.318337
residual sugar          4.540655
chlorides               5.680347
free sulfur dioxide     1.250567
total sulfur dioxide    1.515531
density                 0.071288
pH                      0.193683
sulphates               2.428672
alcohol                 0.860829
quality                 0.217802
dtype: float64
```

#acceptable range is +/-0.5.

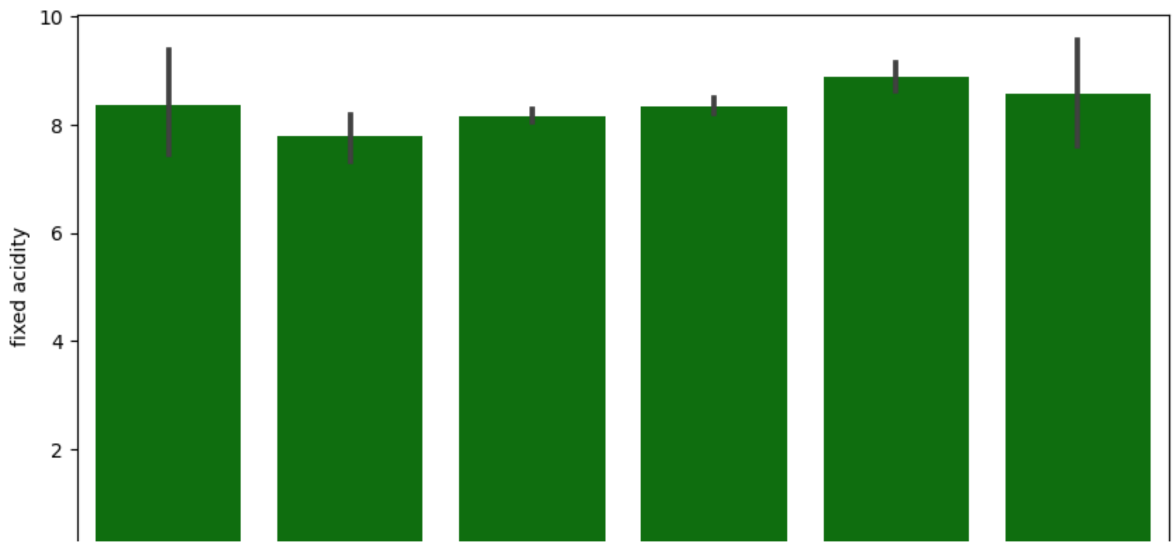
We observe that fixed acidity ,volatile acidity, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, sulphates and alcohol are all outside the acceptable range of +/-0.5. This skewness indicates outliers being present in our dataset that will need to be treated if required.

Visualization

In [13]:
```
plt.figure(figsize=(10,7))
sns.countplot(x='quality', data=df)
plt.xlabel('Quality of Red Wine')
plt.ylabel('Count of rows in the dataset')
plt.show()
```



In [14]:
```
index=0
labels=df['quality']
features=df.drop('quality',axis=1)

for col in features.items():
    plt.figure(figsize=(10,5))
    sns.barplot(x=labels, y=col[index], data=df, color="green")
plt.tight_layout()
plt.show()
```

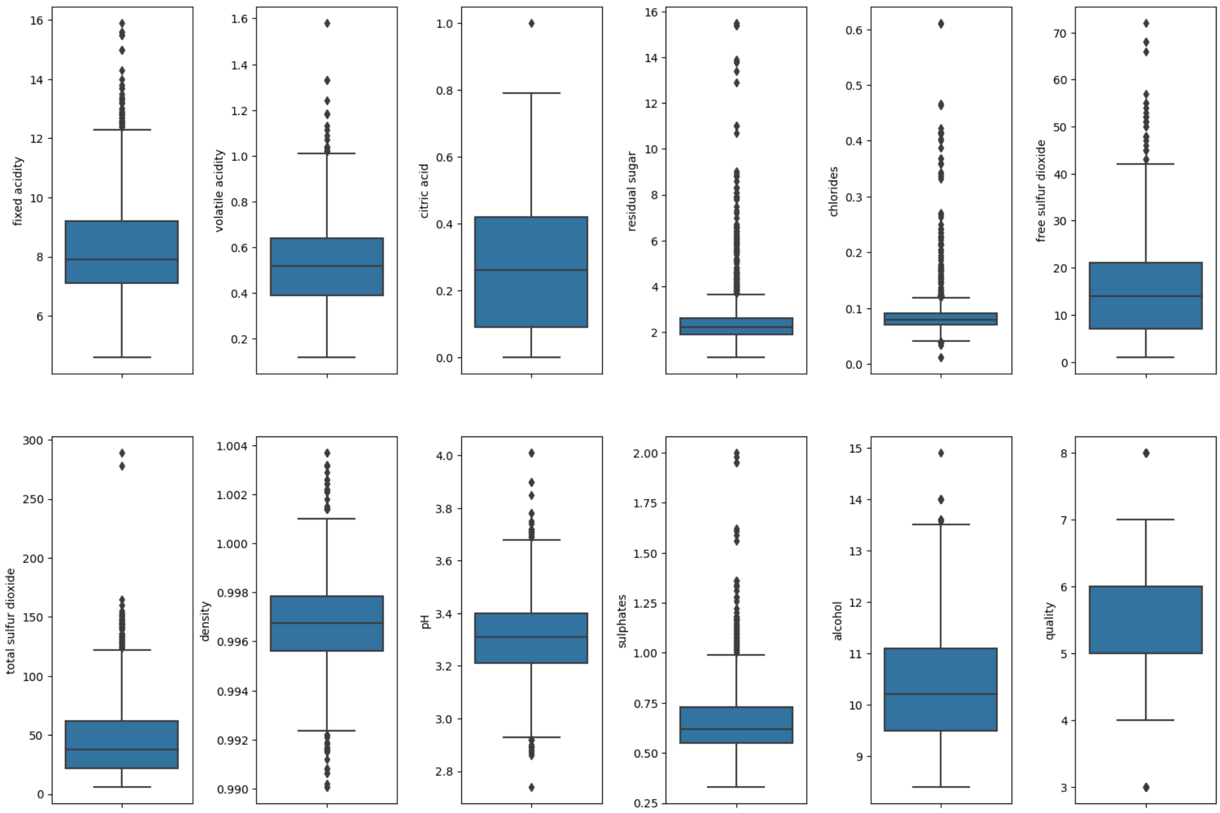


Observation regaring feature compared to the label are:

1. fixed acidity vs quality- no fixed pattern
2. volatile acidity vs quality- there is a decreaing trend
3. citric acid vs quality- there is an increasing trend
4. residual sugar vs quality- no fixed pattern
5. chlorides vs quality- tere is decreasing trend
6. free sulfur dioxide vs quality- no fixed pattern as it is increasing then decreasing
7. total sulfur dioxide vs quality- no fixed pattern as it is increasing then decreasing
8. density vs quality- no pattern at all
9. pH vs quality- no pattern at all
10. sulphates vs quality- there is an increasing trend
11. alcohol vs quality- there is an increasing trend

so we conclude that to get better quality wine citric acid, sulphates and alcohol columns play a major role.

In [15]:
```
fig, ax= plt.subplots(ncols=6, nrows=2, figsize=(15,10))
index=0
ax=ax.flatten()
for col, value in df.items():
    sns.boxplot(y=col, data=df, ax=ax[index])
    index+=1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
plt.show()
```



We are able to see the whisker details and outliers clearly. I am ignoring the continous outlier sections but the outliers that are single values and far away from the whiskers of the boxplot may need to be treated depending upon further analysis. Now I am just trying to retain as much of data which is possible in the dataset.

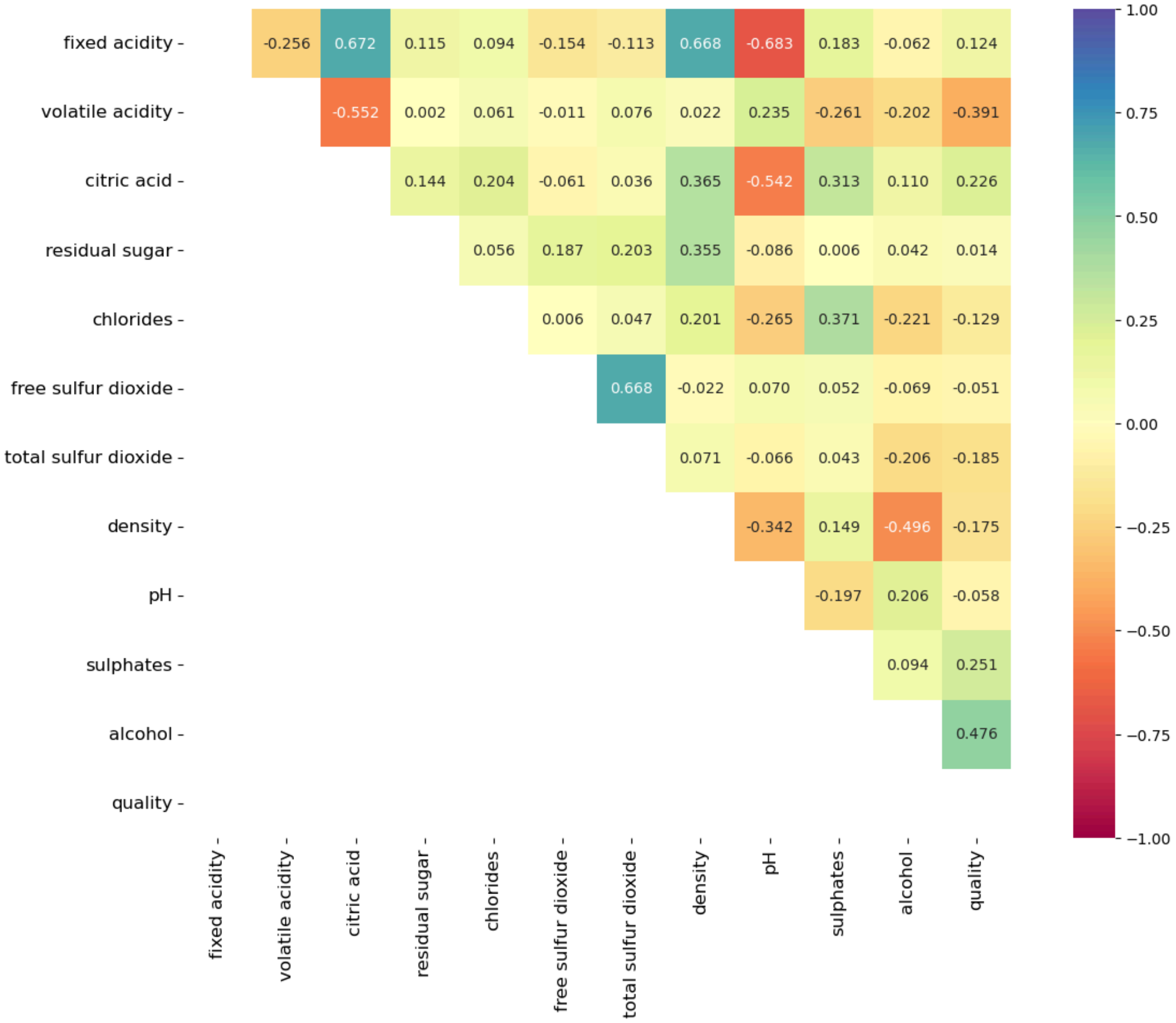
In [16]:
```
fig, ax=plt.subplots(ncols=6, nrows=2, figsize=(15,10))
index= 0
ax =ax.flatten()
for col, value in df.items():
    sns.distplot(value, ax=ax[index], hist=False, color="purple",kde_kws={"shade":True})
    index +=1
plt.tight_layout(pad=0.5,w_pad=0.7, h_pad=5.0)
plt.show()
```

```
C:\Users\Rashmi\AppData\Local\Temp\ipykernel_22140\2845404379.py:5: UserWarning:

`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `kdeplot` (an axes-level function for kernel density plots).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://gist.github.com/mwa
skom/de44147ed2974457ad6372750bbe5751)

  sns.distplot(value, ax=ax[index], hist=False, color="purple",kde_kws={"shade":True})
C:\Users\Rashmi\anaconda3\Lib\site-packages\seaborn\distributions.py:2511: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  kdeplot(**{axis: a}, ax=ax, color=kde_color, **kde_kws)
C:\Users\Rashmi\AppData\Local\Temp\ipykernel_22140\2845404379.py:5: UserWarning:
```

The distribution plots show that few of the columns are in normal distribution category showing a proper bell shape curve. However, we do see skewness in most of the feature columns like citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide,sulphates and alcohol columns. We are going to ignore the label column since it is a categorical column and will need to fix the imbalance data inside it.

With respect to the treatment of skewness and outliers I will perform the removal or treatment after I can see the accuracy dependency of the machine learning models.

In [17]:
```
lower_triangle=np.tril(df.corr())
plt.figure(figsize=(15,10))
sns.heatmap(df.corr(), vmin=-1, vmax=1, annot=True, square=True, fmt='0.3f', annot_kws={'size':10
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.show()
```



Dropping a column

In [18]:
```
df=df.drop('free sulfur dioxide',axis=1)
df
```

Out[18]:

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

1599 rows × 11 columns

Outlier Removal

In [19]:
```
df.shape
```

Out[19]: (1599, 11)

In [20]:
```
#z Score method

z=np.abs(zscore(df))
threshold=3
np.where(z>3)

df=df[(z<3).all(axis=1)]
df
```

Out[20]:

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

1464 rows × 11 columns

I have used the Z score method to get rid of outliers present in our dataset that are not in the acceptable range of +/-0.5 value of skewness.

In [21]:
```
df.shape
```

Out[21]:
```
(1464, 11)
```

Splitting the dataset into 2 variables namely 'X' and 'Y' for feature and label

In [22]:
```
X=df.drop('quality',axis=1)
Y=df['quality']
```

I have bifurcated the dataset into features and labels where X represents all the feature columns and Y represents the target label column.

Taking care of class imbalance

In [23]:
```
Y.value_counts()
```

Out[23]:
```
quality
5    624
6    590
7    187
4     47
8     16
Name: count, dtype: int64
```

In [24]:
```
#adding samples to make all the categorical quality values same

oversample= SMOTE()
X,Y= oversample.fit_resample(X,Y)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[24], line 3
      1 #adding samples to make all the categorical quality values same
----> 3 oversample= SMOTE()
      4 X,Y= oversample.fit_resample(X,Y)

NameError: name 'SMOTE' is not defined
```

In [25]:
```
Y.value_counts()
```

Out[25]:
```
quality
5    624
6    590
7    187
4     47
8     16
Name: count, dtype: int64
```

In [ ]:
```

```

In [26]:
```
Y
```

Out[26]:
```
0       5
1       5
2       5
3       6
4       5
       ..
1594    5
1595    6
1596    6
1597    5
1598    6
Name: quality, Length: 1464, dtype: int64
```

Label Binarization

In [27]:
```
Y=Y.apply(lambda y_value:1 if y_value>=7 else 0)
Y
```

Out[27]:
```
0       0
1       0
2       0
3       0
4       0
       ..
1594    0
1595    0
1596    0
1597    0
1598    0
Name: quality, Length: 1464, dtype: int64
```

In [28]:
```
X
```

Out[28]:

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

1464 rows × 10 columns

Feature Scaling

In [32]:
```
scaler = StandardScaler()
X=pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
X
```

Out[32]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	total sulfur dioxide	density	pH	sulphates	alcohol
0	-0.550028	1.050174	-1.386158	-0.568896	-0.261878	-0.341258	0.636288	1.373350	-0.636986	-0.999199
1	-0.306973	2.117255	-1.386158	0.236748	0.775968	0.769792	0.053575	-0.824088	0.284883	-0.607676
2	-0.306973	1.405868	-1.176549	-0.108528	0.492919	0.332105	0.170117	-0.398777	0.054416	-0.607676
3	1.758994	-1.439680	1.548377	-0.568896	-0.309053	0.534115	0.752831	-1.107628	-0.483341	-0.607676
4	-0.550028	1.050174	-1.386158	-0.568896	-0.261878	-0.341258	0.636288	1.373350	-0.636986	-0.999199
...	...	...	...	...	...	...	...	...	...	...
1459	-1.279193	0.457352	-0.966939	-0.453804	0.398569	-0.004576	-1.053581	0.948040	-0.483341	0.077489
1460	-1.461484	0.160941	-0.862134	-0.223620	-0.922326	0.231101	-0.925384	1.444235	0.899463	0.762654
1461	-1.218429	-0.076188	-0.704927	-0.108528	-0.261878	-0.139249	-0.564102	0.735384	0.822641	0.566893
1462	-1.461484	0.724122	-0.757329	-0.453804	-0.309053	-0.004576	-0.721434	1.798661	0.515351	-0.216153
1463	-1.400721	-1.261833	1.076755	1.387669	-0.686452	-0.071913	-0.709780	0.522729	0.131238	0.566893

1464 rows × 10 columns

Creating the training and testing data sets

In [60]:
```
X_train, X_test, Y_train, Y_test= train_test_split(X,Y,test_size=0.2,random_state=21)
```

Machine Learning Model for Classification and Evaluation Metrics

In [37]:
```
#Classification Model Function

def classify(model, X, Y):
    X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.2,random_state=21)

    #Training the model
    model.fit(X_train, Y_train)

    #Predicting Y_test
    pred = model.predict(X_text)

    #Accuracy Score
    acc_score= (accuracy_score(Y_test,pred))*100
    print("Accuracy Score:", acc_score)

    #Classification Report
    class_report= classification_report(Y_test,pred)
    print("/nClassification Report:/n",class_report)

    #Cross Validation Score
    cv_score=(cross_val_score(model,X,Y,cv=5).mean())*100
    print("Cross Validation Score:", cv_score)

    #Result of accuracy minus cv scores
    result= acc_score- cv_score
    print("/nAccuracy Score - Cross Validation Score is ", result)
```

I have defined a class that will perform the train-test split,training of machine learning model, predicting the label value, getting the accuracy score, generating the clssification report, getting the cross validation score and the result of difference between the accuracy score and cross validation score for any machine learning model that calls for this function.

In [38]:
```python
# Logistic Regression

model=LogisticRegression()
classify(model,X,Y)
```

```
Accuracy Score: 88.73720136518772
/nClassification Report:/n               precision    recall  f1-score   support

           0       0.92      0.96      0.94       251
           1       0.65      0.48      0.55        42

    accuracy                           0.89       293
   macro avg       0.78      0.72      0.74       293
weighted avg       0.88      0.89      0.88       293

Cross Validation Score: 87.09079433353592
/nAccuracy Score - Cross Validation Score is  1.6464070316517905
```

In [42]:
```python
# Support Vector Classifier

model=SVC(C=1.0, kernel='rbf',gamma='auto', random_state=42)
classify(model, X, Y)
```

```
Accuracy Score: 90.10238907849829
/nClassification Report:/n               precision    recall  f1-score   support

           0       0.91      0.98      0.94       251
           1       0.81      0.40      0.54        42

    accuracy                           0.90       293
   macro avg       0.86      0.69      0.74       293
weighted avg       0.89      0.90      0.89       293

Cross Validation Score: 87.29533872551312
/nAccuracy Score - Cross Validation Score is  2.807050352985172
```

In [44]:
```python
# Decision Tree Classifier

model= DecisionTreeClassifier(random_state=21,max_depth=15)
classify(model, X, Y)
```

```
Accuracy Score: 90.10238907849829
/nClassification Report:/n               precision    recall  f1-score   support

           0       0.95      0.94      0.94       251
           1       0.64      0.69      0.67        42

    accuracy                           0.90       293
   macro avg       0.80      0.81      0.80       293
weighted avg       0.90      0.90      0.90       293

Cross Validation Score: 82.10107999438965
/nAccuracy Score - Cross Validation Score is  8.001309084108641
```

In [45]:
```
# Random Forest Classifier

model=RandomForestClassifier(max_depth=15, random_state=111)
classify(model, X, Y)
```

```
Accuracy Score: 91.80887372013652
/nClassification Report:/n               precision    recall  f1-score   support

           0       0.94      0.96      0.95       251
           1       0.75      0.64      0.69        42

    accuracy                           0.92       293
   macro avg       0.85      0.80      0.82       293
weighted avg       0.91      0.92      0.92       293

Cross Validation Score: 87.63710318387956
/nAccuracy Score - Cross Validation Score is  4.1717705362569575
```

In [46]:
```
# K Neighbors Classifier

model= KNeighborsClassifier(n_neighbors=15)
classify(model, X, Y)
```

```
Accuracy Score: 86.3481228668942
/nClassification Report:/n               precision    recall  f1-score   support

           0       0.91      0.93      0.92       251
           1       0.53      0.45      0.49        42

    accuracy                           0.86       293
   macro avg       0.72      0.69      0.70       293
weighted avg       0.86      0.86      0.86       293

Cross Validation Score: 86.74973117022769
/nAccuracy Score - Cross Validation Score is  -0.4016083033334894
```

In [49]:
```
# Extra Trees Classifier

model=ExtraTreesClassifier()
classify(model, X, Y)
```

```
Accuracy Score: 90.44368600682594
/nClassification Report:/n               precision    recall  f1-score   support

           0       0.93      0.96      0.95       251
           1       0.72      0.55      0.62        42

    accuracy                           0.90       293
   macro avg       0.82      0.76      0.78       293
weighted avg       0.90      0.90      0.90       293

Cross Validation Score: 87.15928748422085
/nAccuracy Score - Cross Validation Score is  3.2843985226050876
```

In [ ]:

Hyper parameter tuning on the best ML Model

In [50]:
```
svc_param={'kernel':['poly','sigmoid','rbf'],
           'gamma':['scale','auto'],
           'shrinking':[True,False],
           'probability':[True, False],
           'decision_function_shape':['ovo','ovr'],
           'verbose':[True,False]}
```

In [51]:
```
GSCV = GridSearchCV(SVC(),svc_param,cv=5)
```

In [53]:
```
GSCV.fit(X_train,Y_train)
```

```
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM]
```

Out[53]:

```
▸ GridSearchCV
▸ estimator: SVC
    ▸ SVC
```

In [54]:
```
GSCV.best_params_
```

Out[54]:
```
{'decision_function_shape': 'ovo',
 'gamma': 'scale',
 'kernel': 'rbf',
 'probability': True,
 'shrinking': True,
 'verbose': True}
```

In [61]:
```
Final_Model=SVC(decision_function_shape='ovo', gamma='scale',kernel='rbf',probability=True,random
Classifier=Final_Model.fit(X_train,Y_train)
fmod_pred=Final_Model.predict(X_test)
fmod_acc=(accuracy_score(Y_test,fmod_pred))*100
print("Accuracy score for the Best Model is :", fmod_acc)
```

```
[LibSVM]Accuracy score for the Best Model is : 90.10238907849829
```

I have successfully incorporated the Hyper Parameter Tuning on my Final Model and received the accuracy score for it.

In [ ]:

In [ ]: