In [1]:
```
!pip install imblearn
```

```
Requirement already satisfied: imblearn in c:\users\akansha\anaconda3\lib\site-
packages (0.0)
Requirement already satisfied: imbalanced-learn in c:\users\akansha\anaconda3\l
ib\site-packages (from imblearn) (0.9.1)
Requirement already satisfied: scipy>=1.3.2 in c:\users\akansha\anaconda3\lib\s
ite-packages (from imbalanced-learn->imblearn) (1.7.1)
Requirement already satisfied: scikit-learn>=1.1.0 in c:\users\akansha\anaconda
3\lib\site-packages (from imbalanced-learn->imblearn) (1.1.1)
Requirement already satisfied: numpy>=1.17.3 in c:\users\akansha\anaconda3\lib
\site-packages (from imbalanced-learn->imblearn) (1.20.3)
Requirement already satisfied: joblib>=1.0.0 in c:\users\akansha\anaconda3\lib
\site-packages (from imbalanced-learn->imblearn) (1.1.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\akansha\anacond
a3\lib\site-packages (from imbalanced-learn->imblearn) (2.2.0)
```

In [2]:
```
import pandas as pd
import warnings
warnings.simplefilter('ignore')
import joblib
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import scipy.stats as stats

from scipy.stats import zscore
from imblearn.over_sampling import SMOTE
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
```

In [3]:
```
df=pd.read_csv("https://raw.githubusercontent.com/dsrscientist/DSData/master/wine
```

In [4]:
```
df
```

Out[4]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9
...	...	...	...	...	...	...	...	...	...	...	
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11

1599 rows × 12 columns

In [5]:
```
df.shape
```

Out[5]:
```
(1599, 12)
```

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
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
```

In [7]:
```
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 [8]:
```
df.describe()
```

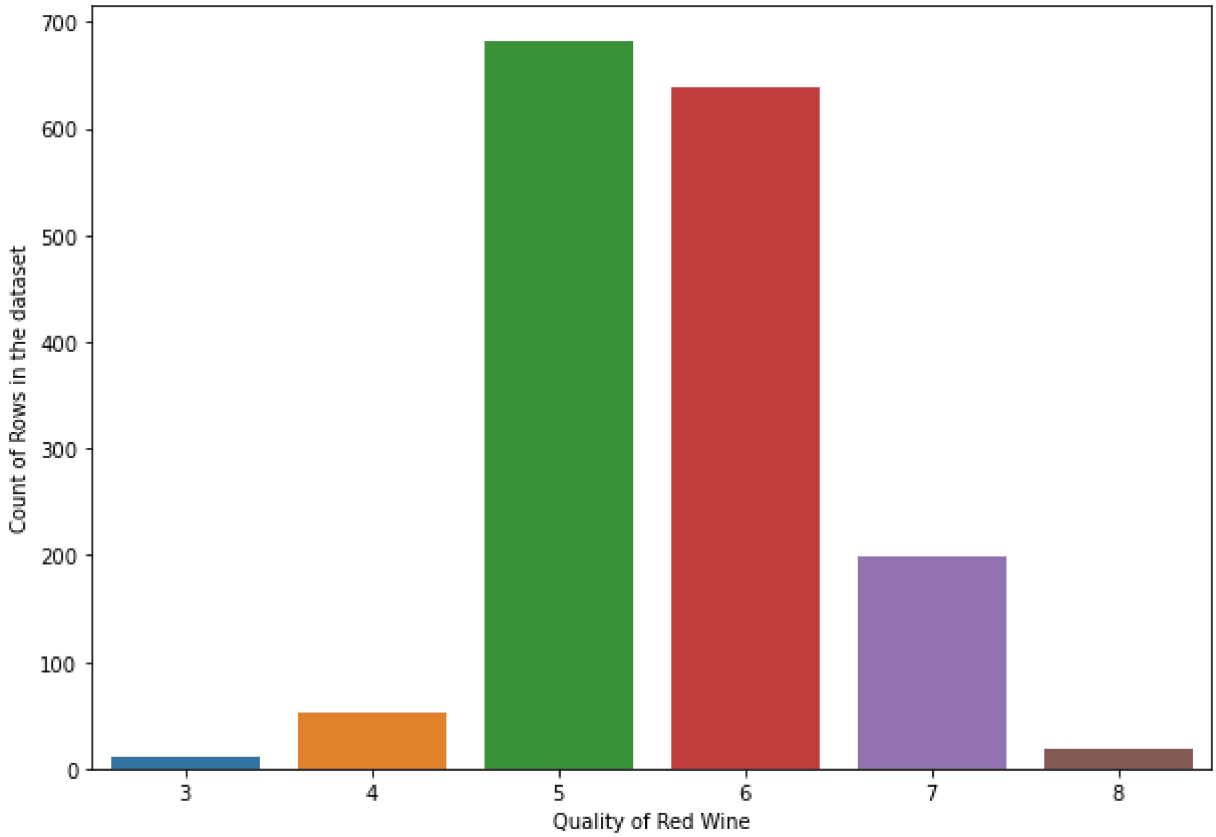
Out[8]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide
count	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
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000

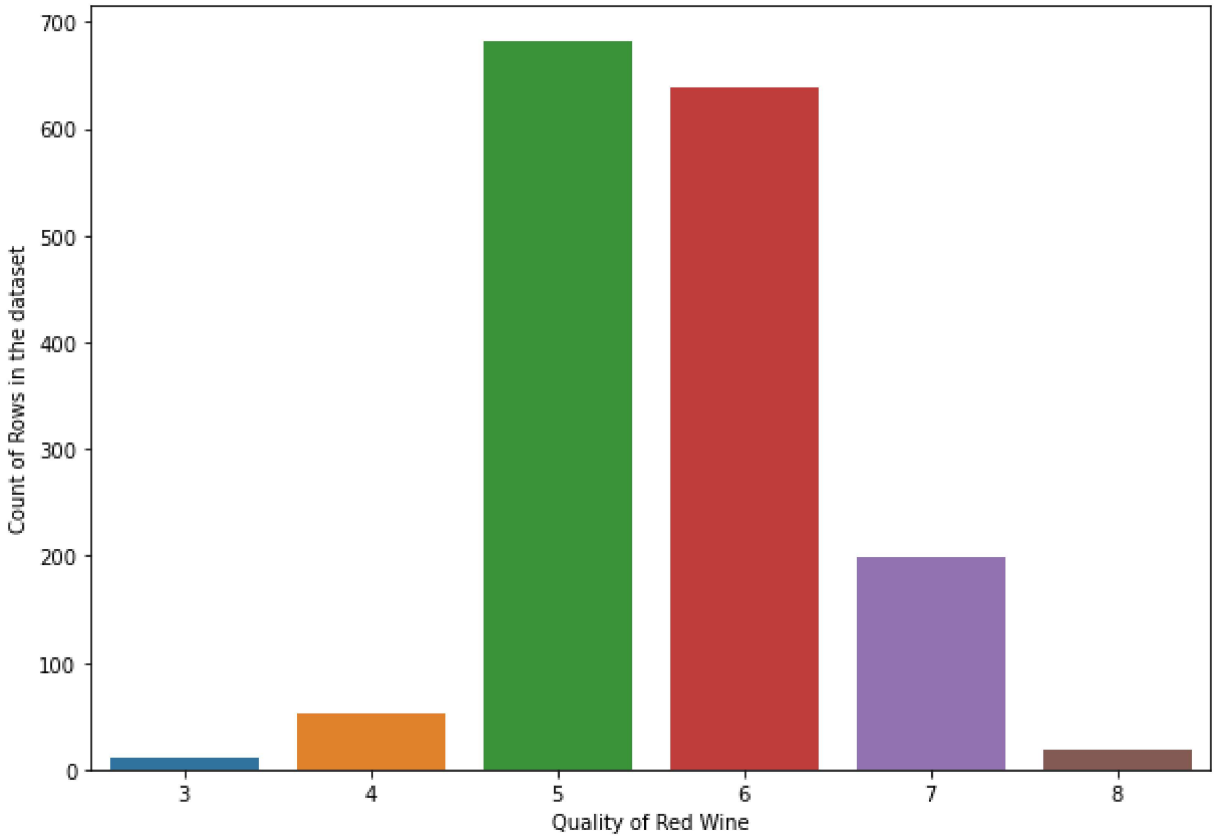
In [9]:
```
df.skew()
```

Out[9]:
```
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
```

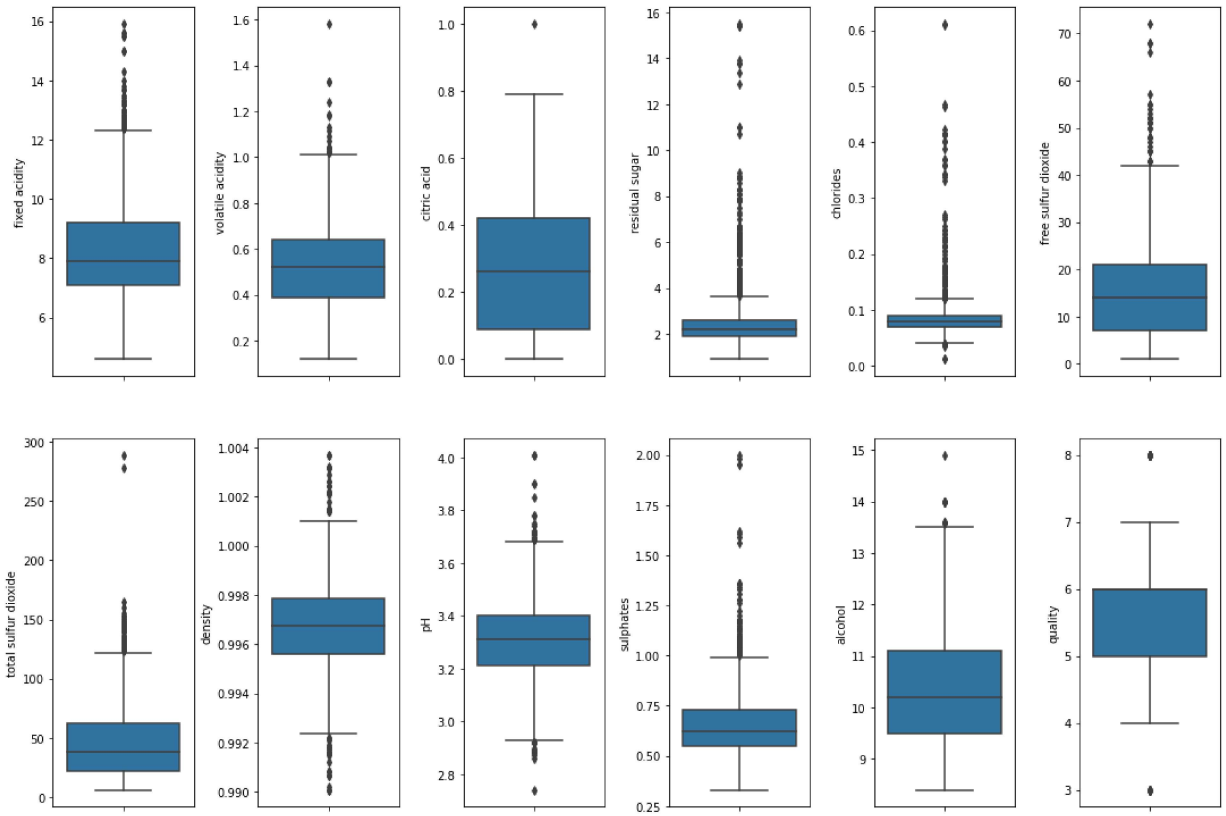
In [10]:
```
plt.figure(figsize=(10,7))
sns.countplot(x ='quality', data = df)
plt.xlabel('Quality of Red Wine')
plt.ylabel('Count of Rows in the dataset')
plt.show()
```



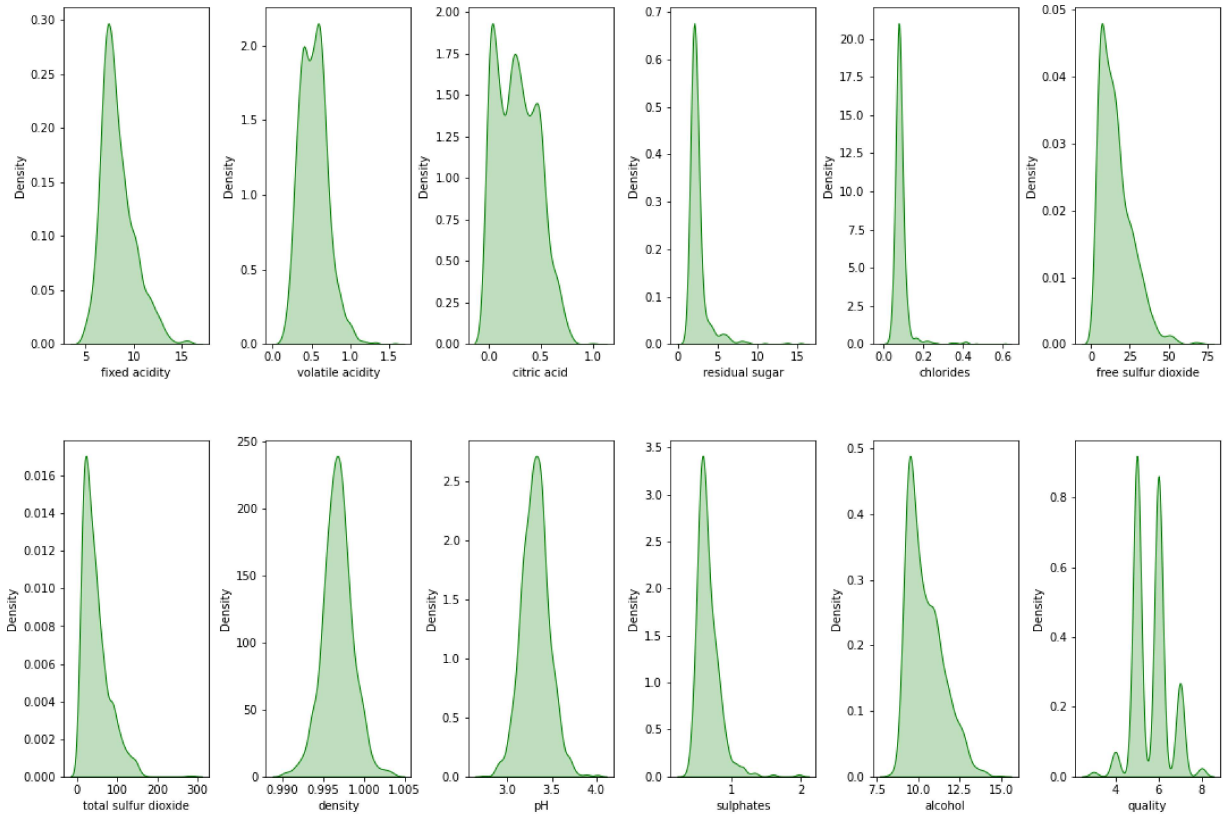
In [11]:
```
plt.figure(figsize=(10,7))
sns.countplot(x ='quality', data = df)
plt.xlabel('Quality of Red Wine')
plt.ylabel('Count of Rows in the dataset')
plt.show()
```



In [12]:
```
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()
```

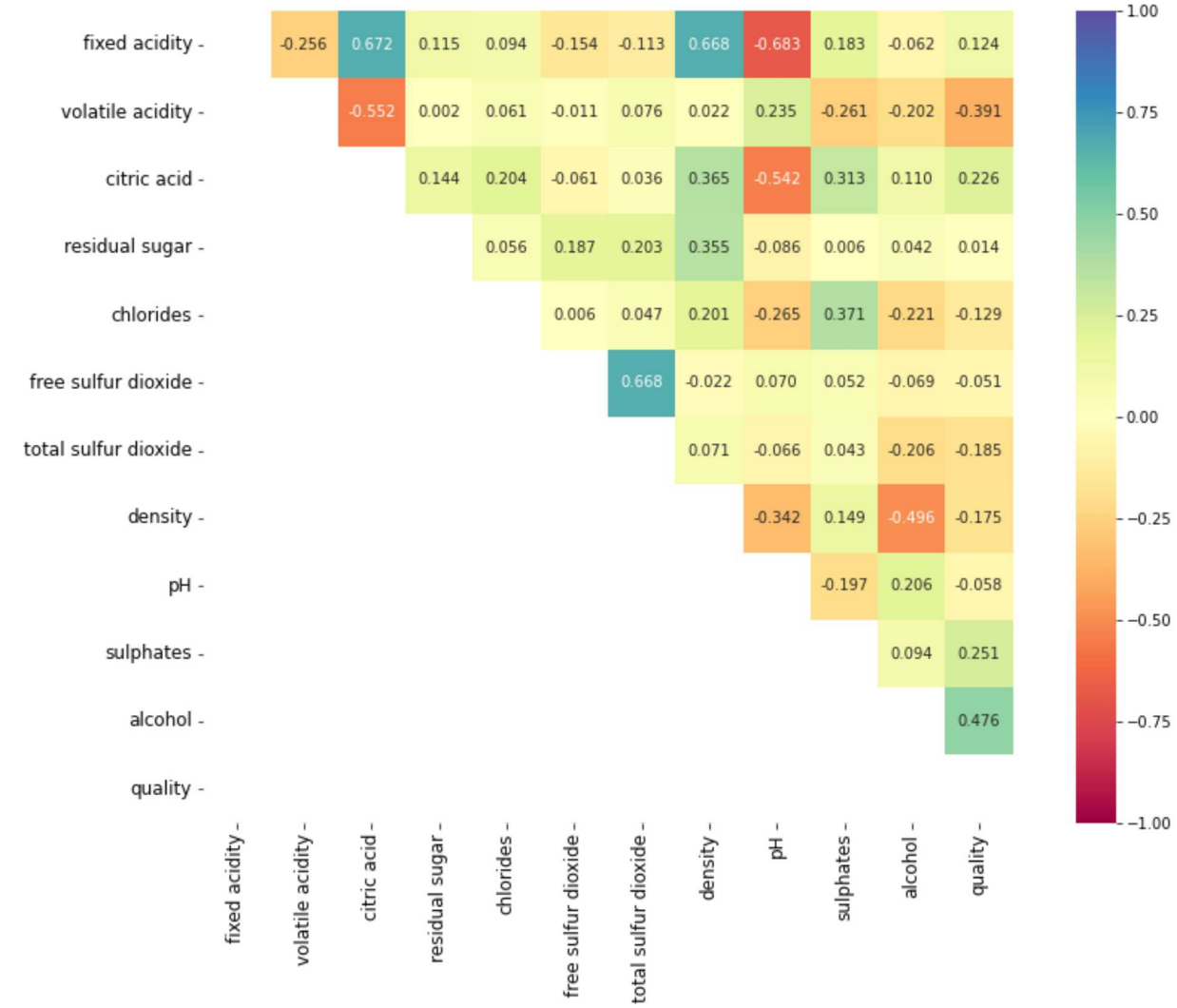


In [13]:
```
fig, ax = plt.subplots(ncols=6, nrows=2, figsize=(15,10))
index = 0
ax = ax.flatten()
for col, value in df.items():
    sns.distplot(value, ax=ax[index], hist=False, color="g", kde_kws={"shade": Tr
    index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
plt.show()
```



Correlation using a Heatmap

In [14]:
```
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}, cmap="Spectral", mask=lower_triangle)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.show()
```



Dropping a column

In [15]:
```
df = df.drop('free sulfur dioxide', axis=1)
df
```

Out[15]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	total sulfur dioxide	density	pH	sulphates	alcohol	qualit
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	

1599 rows × 11 columns

Outlier removal

In [16]:
```
df.shape
```

Out[16]:
```
(1599, 11)
```

In [17]:
```
z=np.abs(zscore(df))
threshold=3
np.where(z>3)

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

Out[17]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	total sulfur dioxide	density	pH	sulphates	alcohol	qualit
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 × 11 columns

In [18]:
```
df.shape
```

Out[18]:
```
(1464, 11)
```

In [19]:
```
data_loss=(1599-1464)/1599*100
data_loss
```

Out[19]:
```
8.442776735459661
```

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

In [20]:
```
X = df.drop('quality', axis=1)
Y = df['quality']
```

In [21]:
```
Y.value_counts()
```

Out[21]:
```
5    624
6    590
7    187
4     47
8     16
Name: quality, dtype: int64
```

In [22]:
```
oversample = SMOTE()
X, Y = oversample.fit_resample(X, Y)
```

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

Out[23]:
```
5    624
6    624
7    624
4    624
8    624
Name: quality, dtype: int64
```

In [24]:
```
Y
```

Out[24]:
```
0       5
1       5
2       5
3       6
4       5
       ..
3115    8
3116    8
3117    8
3118    8
3119    8
Name: quality, Length: 3120, dtype: int64
```

In [25]:
```
Y = Y.apply(lambda y_value:1 if y_value>=7 else 0)
Y
```

Out[25]:
```
0       0
1       0
2       0
3       0
4       0
       ..
3115    1
3116    1
3117    1
3118    1
3119    1
Name: quality, Length: 3120, dtype: int64
```

In [26]:
```
X
```

Out[26]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	total sulfur dioxide	density	pH	sulphates
0	7.400000	0.700000	0.000000	1.900000	0.076000	34.000000	0.997800	3.510000	0.560000
1	7.800000	0.880000	0.000000	2.600000	0.098000	67.000000	0.996800	3.200000	0.680000
2	7.800000	0.760000	0.040000	2.300000	0.092000	54.000000	0.997000	3.260000	0.650000
3	11.200000	0.280000	0.560000	1.900000	0.075000	60.000000	0.998000	3.160000	0.580000
4	7.400000	0.700000	0.000000	1.900000	0.076000	34.000000	0.997800	3.510000	0.560000
...	...	...	...	...	...	...	...	...	...
3115	10.024066	0.589978	0.546159	4.186759	0.082247	18.249451	0.996379	3.212495	0.701258
3116	9.510738	0.304259	0.557444	2.782963	0.079148	16.914817	0.996468	3.150000	0.897000
3117	9.233717	0.355428	0.526743	2.245723	0.075012	16.445723	0.995413	3.183257	0.792516
3118	10.614830	0.343612	0.512966	3.409114	0.070639	15.361226	0.997285	3.167034	0.686197
3119	10.960539	0.586054	0.643769	4.675378	0.083685	18.537098	0.998155	3.218457	0.690000

3120 rows × 10 columns

Feature Scaling

In [27]:
```
scaler = StandardScaler()
X = pd.DataFrame(scaler.fit_transform(X), columns=X.columns)
X
```

Out[27]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	total sulfur dioxide	density	pH	sulpha
0	-0.671582	1.039175	-1.516775	-0.641642	-0.128207	-0.097297	0.849255	1.507011	-0.786
1	-0.417688	2.049946	-1.516775	0.124441	1.111268	1.156188	0.236945	-0.767892	0.117
2	-0.417688	1.376099	-1.310902	-0.203880	0.773229	0.662391	0.359407	-0.327589	-0.108
3	1.740419	-1.319291	1.365452	-0.641642	-0.184546	0.890298	0.971718	-1.061428	-0.636
4	-0.671582	1.039175	-1.516775	-0.641642	-0.128207	-0.097297	0.849255	1.507011	-0.786
...	...	...	...	...	...	...	...	...	
3115	0.994010	0.421358	1.294217	1.860999	0.223762	-0.695572	-0.020545	-0.676203	0.278
3116	0.668182	-1.183066	1.352299	0.324678	0.049160	-0.746267	0.033747	-1.134812	1.754
3117	0.492346	-0.895734	1.194285	-0.263282	-0.183898	-0.764086	-0.612094	-0.890762	0.966
3118	1.368990	-0.962083	1.123376	1.009940	-0.430257	-0.805280	0.534019	-1.009810	0.164
3119	1.588424	0.399323	1.796596	2.395747	0.304792	-0.684646	1.066640	-0.632448	0.193

3120 rows × 10 columns

Creating the training and testing data sets

In [28]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_s
```

Machine Learning Model for Classification and Evaluation Metrics

In [29]:
```
def classify(model, X, Y):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, rand


    model.fit(X_train, Y_train)


    pred = model.predict(X_test)


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


    class_report = classification_report(Y_test, pred)
    print("\nClassification Report:\n", class_report)


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


    result = acc_score - cv_score
    print("\nAccuracy Score - Cross Validation Score is", result)
```

In [30]:
```
model=LogisticRegression()
classify(model, X, Y)
```

```
Accuracy Score: 90.38461538461539

Classification Report:
               precision    recall  f1-score   support

           0       0.94      0.91      0.92       391
           1       0.85      0.90      0.87       233

    accuracy                           0.90       624
   macro avg       0.89      0.90      0.90       624
weighted avg       0.91      0.90      0.90       624

Cross Validation Score: 88.23717948717949

Accuracy Score - Cross Validation Score is 2.147435897435898
```

In [31]:
```
model=SVC(C=1.0, kernel='rbf', gamma='auto', random_state=42)
classify(model, X, Y)
```

```
Accuracy Score: 92.94871794871796

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.94      0.94       391
           1       0.90      0.91      0.91       233

    accuracy                           0.93       624
   macro avg       0.92      0.93      0.92       624
weighted avg       0.93      0.93      0.93       624

Cross Validation Score: 90.41666666666666

Accuracy Score - Cross Validation Score is 2.532051282051299
```

In [32]:
```
model=DecisionTreeClassifier(random_state=21, max_depth=15)
classify(model, X, Y)
```

```
Accuracy Score: 93.10897435897436

Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.93      0.94       391
           1       0.89      0.93      0.91       233

    accuracy                           0.93       624
   macro avg       0.92      0.93      0.93       624
weighted avg       0.93      0.93      0.93       624

Cross Validation Score: 88.97435897435898

Accuracy Score - Cross Validation Score is 4.134615384615387
```

In [33]:
```
model=RandomForestClassifier(max_depth=15, random_state=111)
classify(model, X, Y)
```

```
Accuracy Score: 95.99358974358975

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.96      0.97       391
           1       0.94      0.96      0.95       233

    accuracy                           0.96       624
   macro avg       0.96      0.96      0.96       624
weighted avg       0.96      0.96      0.96       624

Cross Validation Score: 92.5

Accuracy Score - Cross Validation Score is 3.4935897435897516
```

In [34]:
```
model=KNeighborsClassifier(n_neighbors=15)
classify(model, X, Y)
```

```
Accuracy Score: 91.66666666666666

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.91      0.93       391
           1       0.86      0.93      0.89       233

    accuracy                           0.92       624
   macro avg       0.91      0.92      0.91       624
weighted avg       0.92      0.92      0.92       624

Cross Validation Score: 88.62179487179486

Accuracy Score - Cross Validation Score is 3.0448717948717956
```

In [35]:
```
model=ExtraTreesClassifier()
classify(model, X, Y)
```

```
Accuracy Score: 96.47435897435898

Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.96      0.97       391
           1       0.93      0.97      0.95       233

    accuracy                           0.96       624
   macro avg       0.96      0.97      0.96       624
weighted avg       0.97      0.96      0.96       624

Cross Validation Score: 93.71794871794872

Accuracy Score - Cross Validation Score is 2.7564102564102626
```

In [37]:
```
svc_param = {'kernel' : ['poly', 'sigmoid', 'rbf'],
             'gamma' : ['scale', 'auto'],
             'shrinking' : [True, False],
             'random_state' : [21,42,104],
             'probability' : [True, False],
             'decision_function_shape' : ['ovo', 'ovr'],
             'verbose' : [True, False]}
```

In [38]:
```
GSCV = GridSearchCV(SVC(), svc_param, cv=5)
```

In [39]:
```
GSCV.fit(X_train,Y_train)
```

```
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibS
VM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][L
ibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSV
M][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][Li
bSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibS
VM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][L
ibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSV
M][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][Li
bSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibS
VM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][L
ibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSV
M][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][Li
bSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibS
VM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][L
ibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSV
M][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][Li
bSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibS
VM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][L
ibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSV
M][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][Li
bSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibS
VM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][L
ibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSV
M][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][Li
bSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibS
VM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][L
ibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSV
M][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][Li
bSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibS
VM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][L
ibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSV
M][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][Li
bSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibS
VM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][L
ibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSV
M][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][Li
bSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibS
VM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][L
ibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSV
M][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][Li
bSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibS
VM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][L
ibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSV
M][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][Li
bSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
```

```
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibS
VM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][L
ibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSV
M][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][Li
bSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibS
VM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][L
ibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSV
M][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][Li
bSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibS
VM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][L
ibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSV
M][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][Li
bSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibS
VM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][L
ibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSV
M][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][Li
bSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM][LibSVM]
[LibSVM]
```

Out[39]:

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

In [40]:

```python
GSCV.best_params_
```

Out[40]:

```
{'decision_function_shape': 'ovo',
 'gamma': 'auto',
 'kernel': 'rbf',
 'probability': True,
 'random_state': 21,
 'shrinking': True,
 'verbose': True}
```

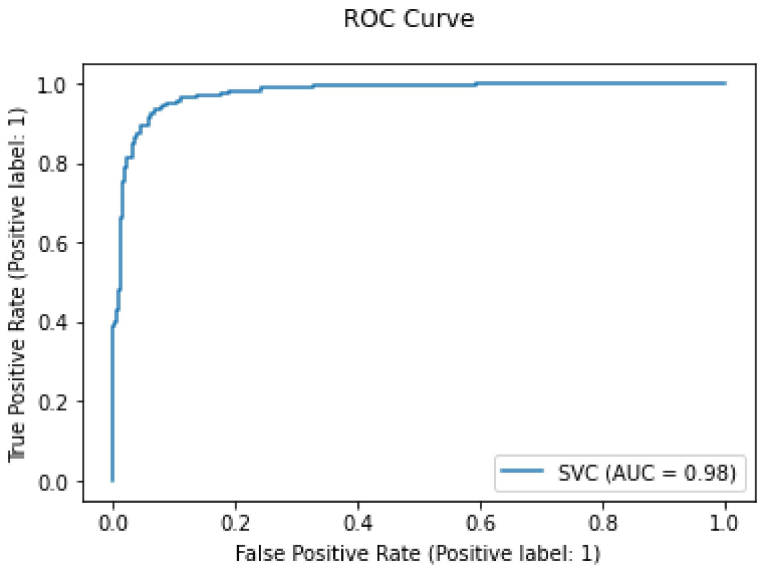
In [41]:

```python
Final_Model = SVC(decision_function_shape='ovo', gamma='scale', kernel='rbf', pro
                  shrinking=True, verbose=True)
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: 93.10897435897436
```

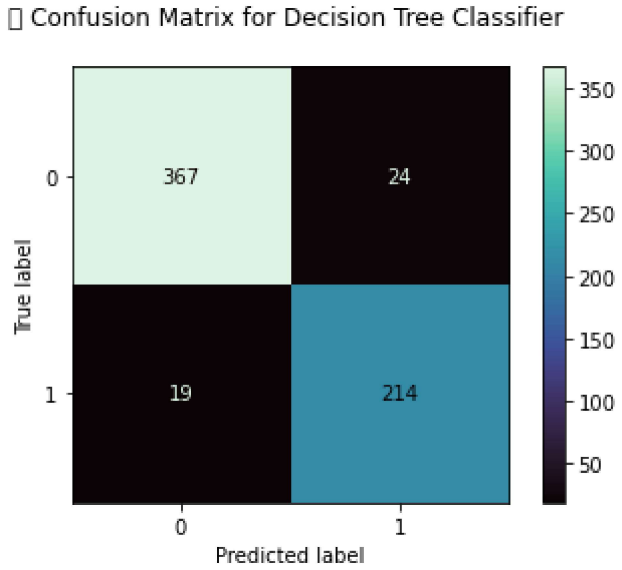
AUC ROC Curve

In [42]:
```
disp = metrics.plot_roc_curve(Final_Model, X_test, Y_test)
disp.figure_.suptitle("ROC Curve")
plt.show()
```



Confusion Matrix

In [43]:
```
class_names = df.columns
metrics.plot_confusion_matrix(Classifier, X_test, Y_test, cmap='mako')
plt.title('\t Confusion Matrix for Decision Tree Classifier \n')
plt.show()
```



In [44]:
```
filename = "FinalModel_1.pkl"
joblib.dump(Final_Model, filename)
```

Out[44]:
```
['FinalModel_1.pkl']
```

In [ ]:
```

```