In [55]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn import svm
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import StandardScaler,LabelEncoder
from sklearn.metrics import accuracy_score
from sklearn.ensemble import VotingClassifier
```

In [3]:
```
wine=pd.read_csv('C:\\Users\\LENOVO\\Documents\\wine.csv')
```

In [4]:
```
# view the first 10 of our dataset
wine.head(10)
```

Out[4]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	9.4
6	7.9	0.60	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	9.4
7	7.3	0.65	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	10.0
8	7.8	0.58	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	9.5
9	7.5	0.50	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	10.5

In [5]:
```
# the last 5 from our dataset
wine.tail()
```

Out[5]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
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

In [6]:
```
wine.shape
```

Out[6]: (1599, 12)

In [7]:
```
#statistical decription of the dataset
wine.describe()
```

Out[7]:

	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 [8]:
```
wine.isnull()
```

Out[8]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcol
0	False	False	False	False	False	False	False	False	False	False	Fa
1	False	False	False	False	False	False	False	False	False	False	Fa
2	False	False	False	False	False	False	False	False	False	False	Fa
3	False	False	False	False	False	False	False	False	False	False	Fa
4	False	False	False	False	False	False	False	False	False	False	Fa
...	...	...	...	...	...	...	...	...	...	...	
1594	False	False	False	False	False	False	False	False	False	False	Fa
1595	False	False	False	False	False	False	False	False	False	False	Fa
1596	False	False	False	False	False	False	False	False	False	False	Fa
1597	False	False	False	False	False	False	False	False	False	False	Fa
1598	False	False	False	False	False	False	False	False	False	False	Fa

1599 rows × 12 columns

In [9]:
```
wine.columns
```

Out[9]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

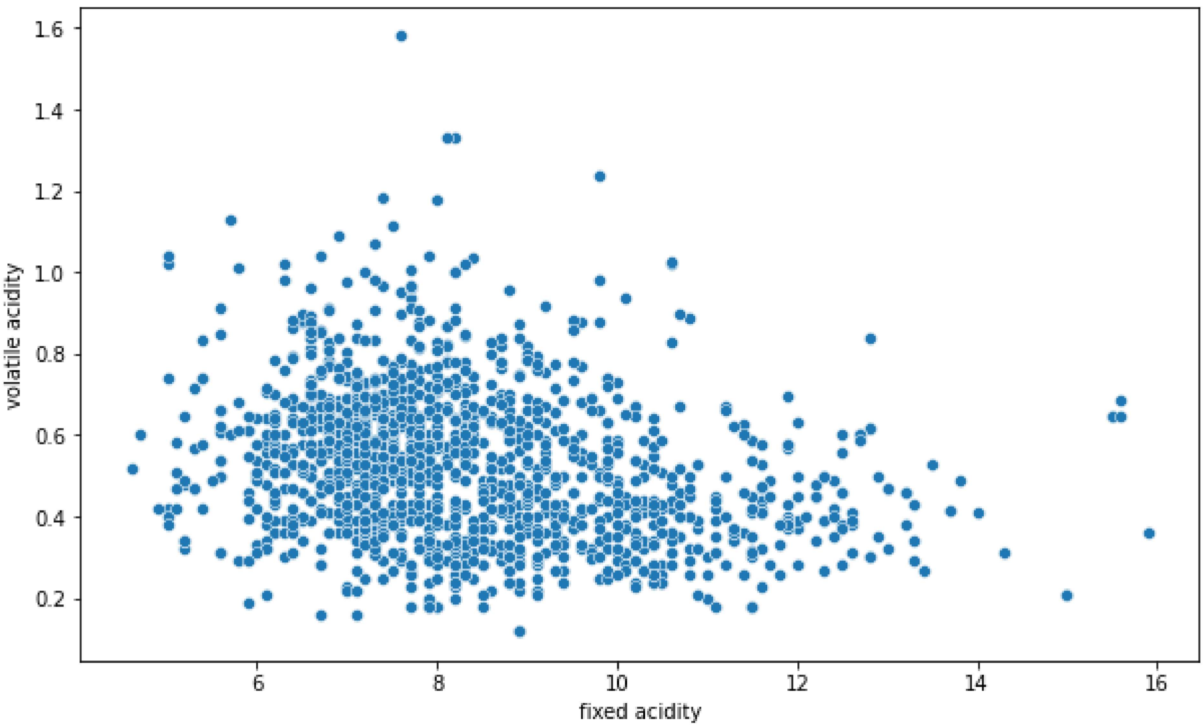
In [10]:
```
wine.nunique
```

Out[10]:
```
<bound method DataFrame.nunique of       fixed acidity  volatile acidity  citri
c acid  residual sugar  chlorides  \
0               7.4             0.700         0.00             1.9      0.076
1               7.8             0.880         0.00             2.6      0.098
2               7.8             0.760         0.04             2.3      0.092
3              11.2             0.280         0.56             1.9      0.075
4               7.4             0.700         0.00             1.9      0.076
...             ...               ...          ...             ...        ...
1594            6.2             0.600         0.08             2.0      0.090
1595            5.9             0.550         0.10             2.2      0.062
1596            6.3             0.510         0.13             2.3      0.076
1597            5.9             0.645         0.12             2.0      0.075
1598            6.0             0.310         0.47             3.6      0.067

      free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                    11.0                  34.0  0.99780  3.51       0.56
1                    25.0                  67.0  0.99680  3.20       0.68
2                    15.0                  54.0  0.99700  3.26       0.65
3                    17.0                  60.0  0.99800  3.16       0.58
4                    11.0                  34.0  0.99780  3.51       0.56
...                   ...                   ...      ...   ...        ...
1594                 32.0                  44.0  0.99490  3.45       0.58
1595                 39.0                  51.0  0.99512  3.52       0.76
1596                 29.0                  40.0  0.99574  3.42       0.75
1597                 32.0                  44.0  0.99547  3.57       0.71
1598                 18.0                  42.0  0.99549  3.39       0.66

      alcohol  quality
0         9.4        5
1         9.8        5
2         9.8        5
3         9.8        6
4         9.4        5
...       ...      ...
1594     10.5        5
1595     11.2        6
1596     11.0        6
1597     10.2        5
1598     11.0        6

[1599 rows x 12 columns]>
```

In [11]:
```
wine['quality'].value_counts()
```

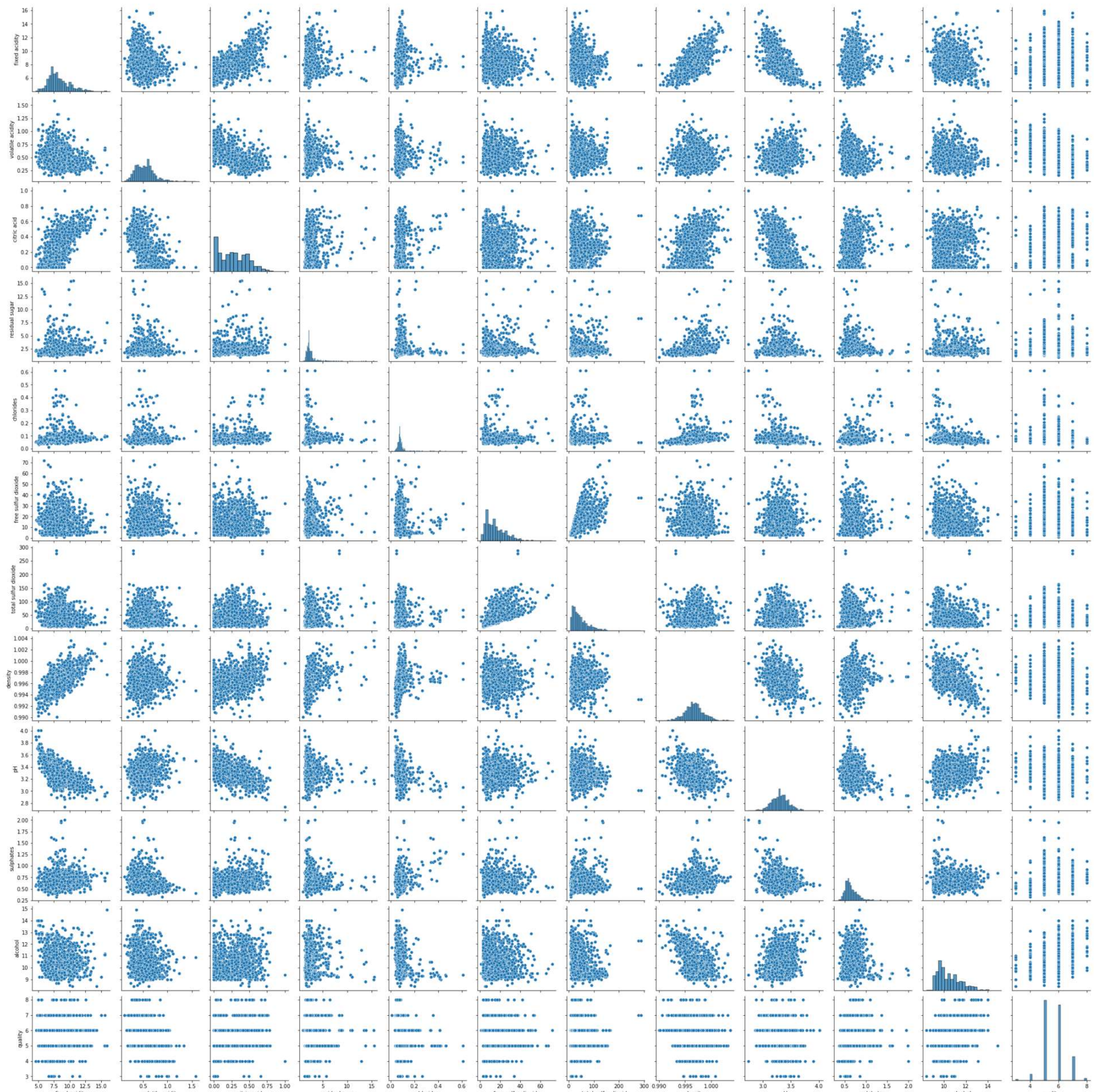
Out[11]:
```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

In [12]:
```
plt.figure(figsize=(10,6))
sns.scatterplot(x=wine['fixed acidity'],y=wine['volatile acidity']);
```



In [13]:
```
plt.figure(figsize=(10,6))
sns.pairplot(wine);
```

```
<Figure size 720x432 with 0 Axes>
```



In [14]:
```
# Preprocessing the dataset
bins=(1, 6.5, 8)
group_name=['good','bad']
wine['quality']= pd.cut(wine['quality'],bins=bins,labels=group_name)
```

In [15]:
```
wine['quality'].unique()
```

Out[15]:
```
['good', 'bad']
Categories (2, object): ['good' < 'bad']
```

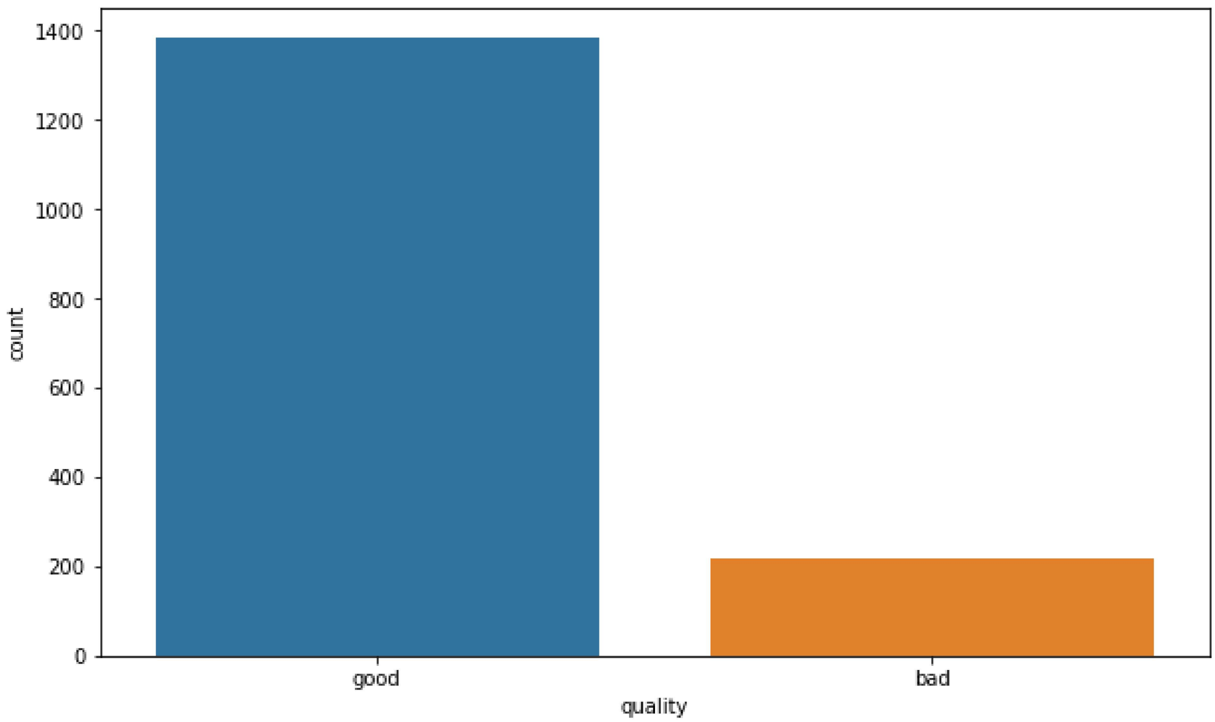
In [19]:
```
wine[:10]
```

Out[19]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	9.4
6	7.9	0.60	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	9.4
7	7.3	0.65	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	10.0
8	7.8	0.58	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	9.5
9	7.5	0.50	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	10.5

In [16]:
```
plt.figure(figsize=(10,6))
sns.countplot(wine['quality']);
```

```
C:\Users\LENOVO\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWa
rning: Pass the following variable as a keyword arg: x. From version 0.12, the
only valid positional argument will be `data`, and passing other arguments with
out an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



In [26]:
```
label_quality=LabelEncoder()
```

In [28]:
```
wine['quality'] = label_quality.fit_transform(wine['quality'])
```

In [29]:
```
wine[:10]
```

Out[29]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	9.4
6	7.9	0.60	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	9.4
7	7.3	0.65	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	10.0
8	7.8	0.58	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	9.5
9	7.5	0.50	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	10.5

In [31]:
```
wine['quality'].value_counts()
```

Out[31]:
```
1    1382
0     217
Name: quality, dtype: int64
```

In [32]:
```
x=wine.drop(['quality'],axis=1)
```

In [33]:
```
x
```

Out[33]:

	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 × 11 columns

In [34]:
```
y=wine['quality']
```

In [36]:
```
y
```

Out[36]:
```
0       1
1       1
2       1
3       1
4       1
       ..
1594    1
1595    1
1596    1
1597    1
1598    1
Name: quality, Length: 1599, dtype: int32
```

In [37]:
```
# training , testing and spliting the dataset
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=4
```

In [40]:
```
# applying standard scaler to get optimizer result
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test= sc.transform(x_test)
```

In [41]:
```
x_train
```

Out[41]:
```
array([[-1.09852129,  1.50992721, -1.38824517, ...,  1.14517889,
        -0.8026406 ,  1.10997762],
       [-1.09852129,  0.6327804 , -0.30279894, ...,  1.79318941,
         0.02216045, -0.59560931],
       [ 1.33427414,  1.70799262,  0.52420772, ..., -0.99325583,
        -0.8026406 , -0.50085448],
       ...,
       [-0.75097909,  0.18005946, -0.71630226, ...,  0.17316311,
        -0.39024008, -1.16413829],
       [ 0.29164752, -0.046301  ,  0.36914398, ..., -0.34524531,
        -0.27241136, -0.8798738 ],
       [ 1.21842673, -1.51764404,  1.0927748 , ...,  0.10836206,
         1.25936202,  0.44669381]])
```

In [42]:
```
#USING DECISION TREE CLASSIFIER ALGORITHM
dtc=DecisionTreeClassifier(random_state=45)
```

In [43]:
```
dtc.fit(x_train,y_train)
```

Out[43]:
```
DecisionTreeClassifier(random_state=45)
```

In [44]:
```
dtc_pred= dtc.predict(x_test)
```

In [48]:
```
dtc_pred[:30]
```

Out[48]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 0, 1, 1])
```

In [58]:
```
# to determine the accuracy of the model
accuracy_score(y_test,dtc_pred)*100
```

Out[58]:
```
86.0
```

In [50]:
```
#USING RANDOM FOREST CLASSIFIER ALGORITHM
rfc=RandomForestClassifier(n_estimators=200)
```

In [51]:
```
rfc.fit(x_train,y_train)
```

Out[51]:
```
RandomForestClassifier(n_estimators=200)
```

In [52]:
```
rfc_pred = rfc.predict(x_test)
```

In [53]:
```
rfc_pred[:30]
```

Out[53]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1])
```

In [57]:
```
# to determine the accuracy of the model
accuracy_score(y_test,rfc_pred)*100
```

Out[57]:
```
89.5
```

In [59]:
```
#USING KNEAREST NIEGHBOR  ALGORITHM
knn=KNeighborsClassifier(n_neighbors=5)
```

In [60]:
```
knn.fit(x_train,y_train)
```

Out[60]:
```
KNeighborsClassifier()
```

In [61]:
```
knn_pred= knn.predict(x_test)
```

In [63]:
```
knn_pred[:30]
```

Out[63]:
```
array([1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1])
```

In [64]:
```
# to determine the accuracy of the model
accuracy_score(y_test,rfc_pred)*100
```

Out[64]:
```
89.5
```

In [66]:
```
nb=GaussianNB()
```

In [67]:
```
nb.fit(x_train,y_train)
```

Out[67]:
```
GaussianNB()
```

In [68]:
```
nb_pred=nb.predict(x_test)
```

In [70]:
```
nb_pred[:30]
```

Out[70]:
```
array([1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1])
```

In [71]:
```
# to determine the accuracy of the model
accuracy_score(y_test,nb_pred)*100
```

Out[71]:
```
82.0
```

In [74]:
```
# USING VOTING ENSEMBLE TO DECIDE THE BEST MODEL
estimators=[('Decision',dtc), ('Random',rfc),('KNeighbors',knn),('Gaussian',nb)]
```

In [75]:
```
VC= VotingClassifier(estimators=estimators,voting='hard')
```

In [76]:
```
VC.fit(x_train,y_train)
```

Out[76]:
```
VotingClassifier(estimators=[('Decision',
                              DecisionTreeClassifier(random_state=45)),
                             ('Random',
                              RandomForestClassifier(n_estimators=200)),
                             ('KNeighbors', KNeighborsClassifier()),
                             ('Gaussian', GaussianNB())])
```

In [77]:
```
vc_pred=VC.predict(x_test)
```

In [78]:
```
vc_pred[:30]
```

Out[78]:
```
array([1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1])
```

In [79]:
```
# to determine the accuracy of the model
accuracy_score(y_test,vc_pred)*100
```

Out[79]:
```
88.0
```

In [80]:
```
#USING BAGGING ENSEMBLE
from sklearn.ensemble import BaggingClassifier
```

In [81]:
```
BC= BaggingClassifier(base_estimator=dtc,n_estimators=10)
```

In [82]:
```
BC.fit(x_train,y_train)
```

Out[82]:
```
BaggingClassifier(base_estimator=DecisionTreeClassifier(random_state=45))
```

In [83]:
```
bc_pred=BC.predict(x_test)
```

In [84]:
```
bc_pred[:30]
```

Out[84]:
```
array([0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1])
```

In [85]:
```
# to determine the accuracy of the model
accuracy_score(y_test,bc_pred)*100
```

Out[85]:
```
87.5
```

In [86]:
```
# USING STACKING ENSEMBLE
from sklearn.ensemble import StackingClassifier
```

In [87]:
```
SR= StackingClassifier(estimators=estimators,final_estimator=dtc)
```

In [88]:
```
SR.fit(x_train,y_train)
```

Out[88]:
```
StackingClassifier(estimators=[('Decision',
                                DecisionTreeClassifier(random_state=45)),
                               ('Random',
                                RandomForestClassifier(n_estimators=200)),
                               ('KNeighbors', KNeighborsClassifier()),
                               ('Gaussian', GaussianNB())],
                   final_estimator=DecisionTreeClassifier(random_state=45))
```

In [89]:
```
sr_pred=SR.predict(x_test)
```

In [90]:
```
sr_pred[:30]
```

Out[90]:
```
array([1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1])
```

In [91]:
```
# to determine the accuracy of the model
accuracy_score(y_test,sr_pred)*100
```

Out[91]:
```
87.5
```

In [92]:
```
# USING VOTING ENSEMBLE TO DECIDE THE BEST MODEL
from sklearn.ensemble import VotingClassifier
```

In [93]:
```
estimators=[('Voting',VC), ('Bagging',BC),('Stacking',SR)]
```

In [94]:
```
VC= VotingClassifier(estimators=estimators,voting='hard')
```

In [95]:
```
VC.fit(x_train,y_train)
```

Out[95]:
```
VotingClassifier(estimators=[('Voting',
                              VotingClassifier(estimators=[('Decision',
                                                            DecisionTreeClassif
ier(random_state=45)),
                                                           ('Random',
                                                            RandomForestClassif
ier(n_estimators=200)),
                                                           ('KNeighbors',
                                                            KNeighborsClassifie
r()),
                                                           ('Gaussian',
                                                            GaussianNB())])),
                             ('Bagging',
                              BaggingClassifier(base_estimator=DecisionTreeClas
sifier(random_state=45))),
                             ('Stacking',
                              StackingClassifier(estimators=[('Decision',
                                                              DecisionTreeClass
ifier(random_state=45)),
                                                             ('Random',
                                                              RandomForestClass
ifier(n_estimators=200)),
                                                             ('KNeighbors',
                                                              KNeighborsClassif
ier()),
                                                             ('Gaussian',
                                                              GaussianNB())],
                                                 final_estimator=DecisionTreeCl
assifier(random_state=45)))])
```

In [96]:
```
final_pred=VC.predict(x_test)
```

In [97]:
```
final_pred[:30]
```

Out[97]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1])
```

In [98]:
```
# to determine the accuracy of the model
accuracy_score(y_test,final_pred)*100
```

Out[98]:
```
88.75
```

In [100]:
```
# pickle to save model
import pickle
```

In [101]:
```
with open('wine.plk','wb') as f:
    pickle.dump(wine, f, protocol=pickle.HIGHEST_PROTOCOL)
```

In [ ]:
```

```