In [1]:
```python
# Ignore  the warnings
import warnings
warnings.filterwarnings('always')
warnings.filterwarnings('ignore')

# data visualisation and manipulation
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns
import missingno as msno
#configure
# sets matplotlib to inline and displays graphs below the corressponding cell.
%matplotlib inline
style.use('fivethirtyeight')
sns.set(style='whitegrid',color_codes=True)

#import the necessary modelling algos.

#classifiaction.
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC,SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier,AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB


#regression
from sklearn.linear_model import LinearRegression,Ridge,Lasso,RidgeCV
from sklearn.ensemble import RandomForestRegressor,BaggingRegressor,GradientBoostingRegressor,AdaBoostRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor

#model selection
from sklearn.model_selection import train_test_split,cross_validate
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV

#preprocessing
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler,StandardScaler,LabelEncoder

#evaluation metrics
from sklearn.metrics import mean_squared_log_error,mean_squared_error, r2_score,mean_absolute_error # for regre
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score  # for classification
```

In [2]:
```python
df=pd.read_csv('winequality.csv')
```

In [3]:
```python
df.shape
```

Out[3]:
```
(1599, 12)
```

In [4]:
```python
#checking columns
df.columns
```

Out[4]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

In [5]:
```python
df.info()
```

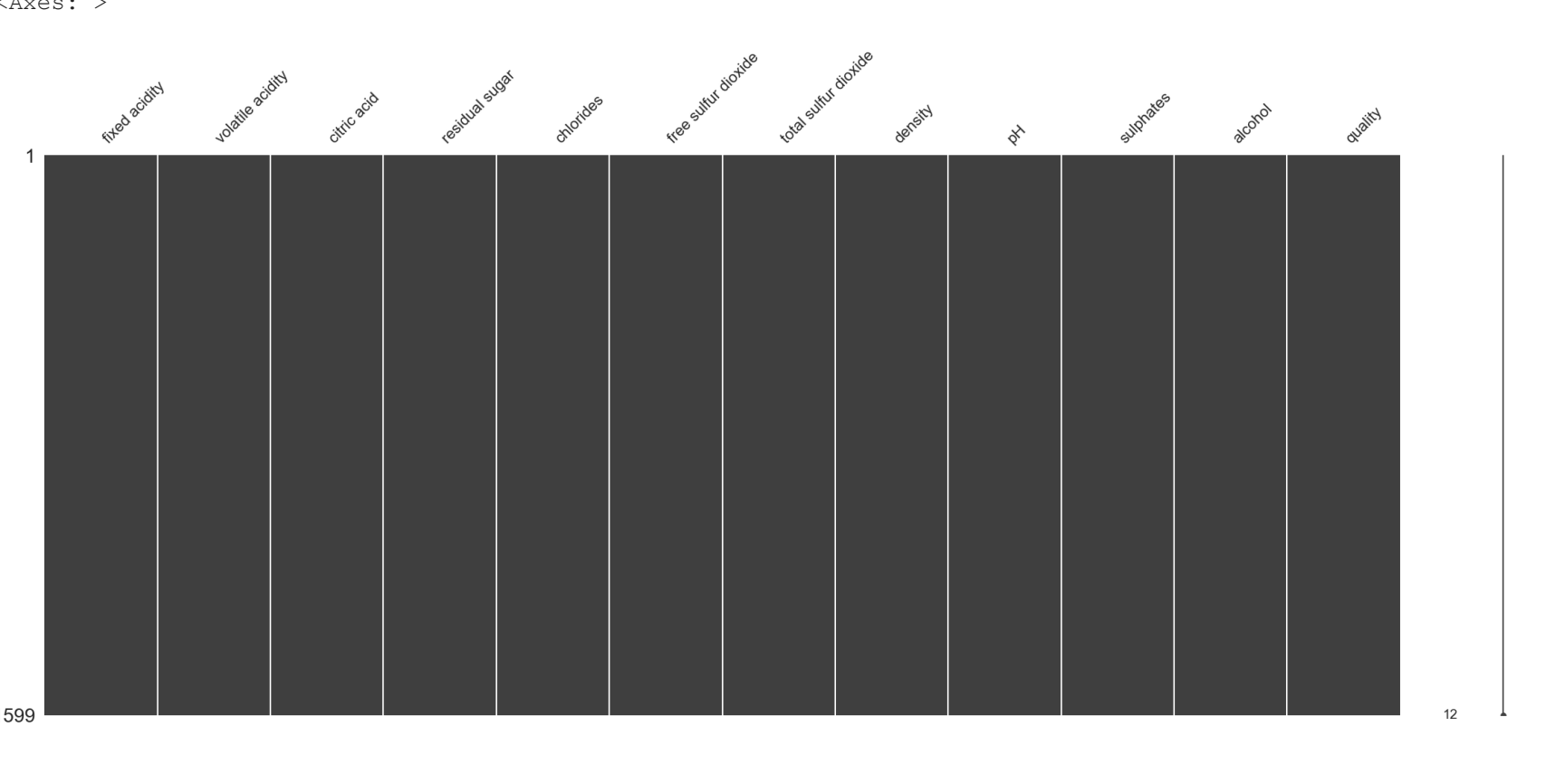
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   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 [6]:
```python
#checking null
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]:
```python
msno.matrix(df)
```

Out[7]:
```
<Axes: >
```



UNIVARIATE ANALYSIS.

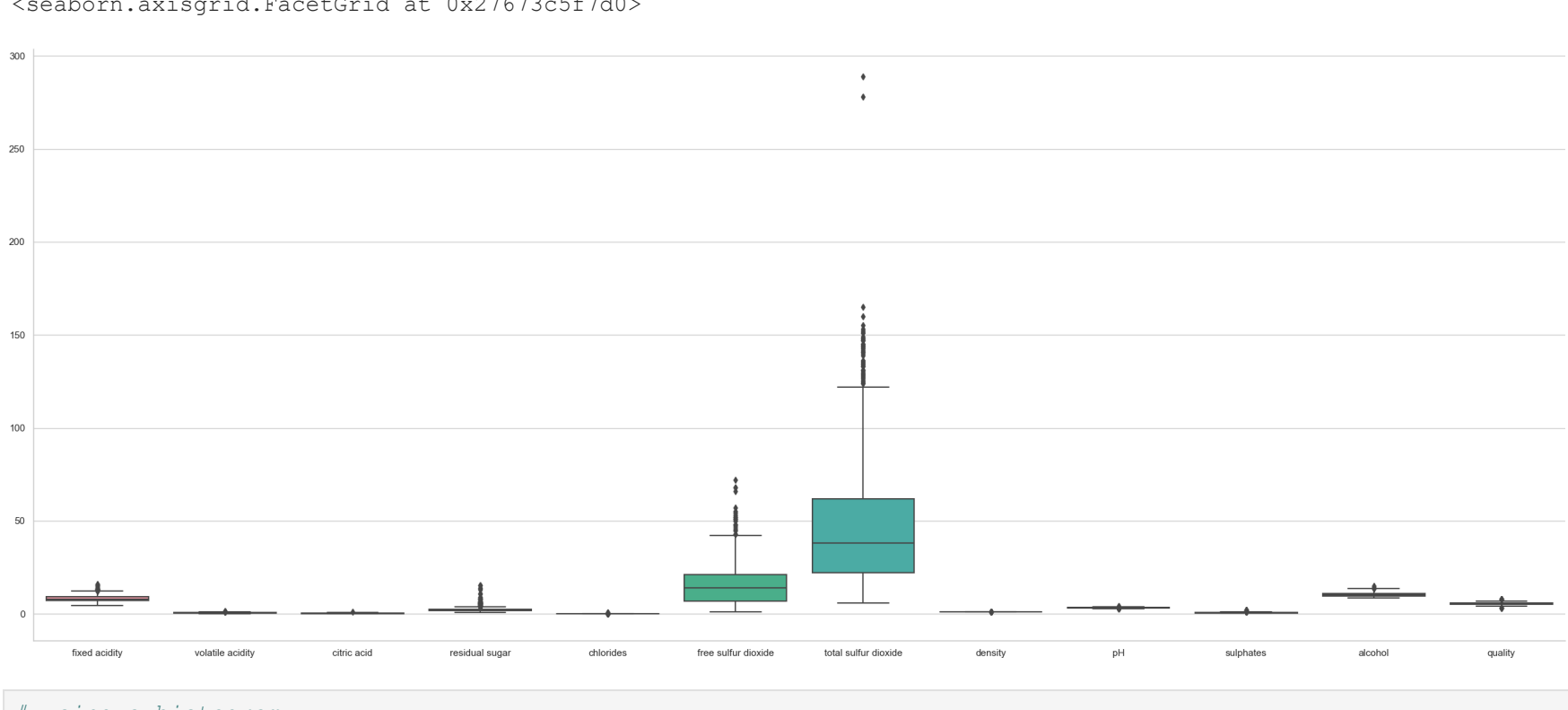
In [8]:
```python
df.describe(include='all')
```

Out[8]:

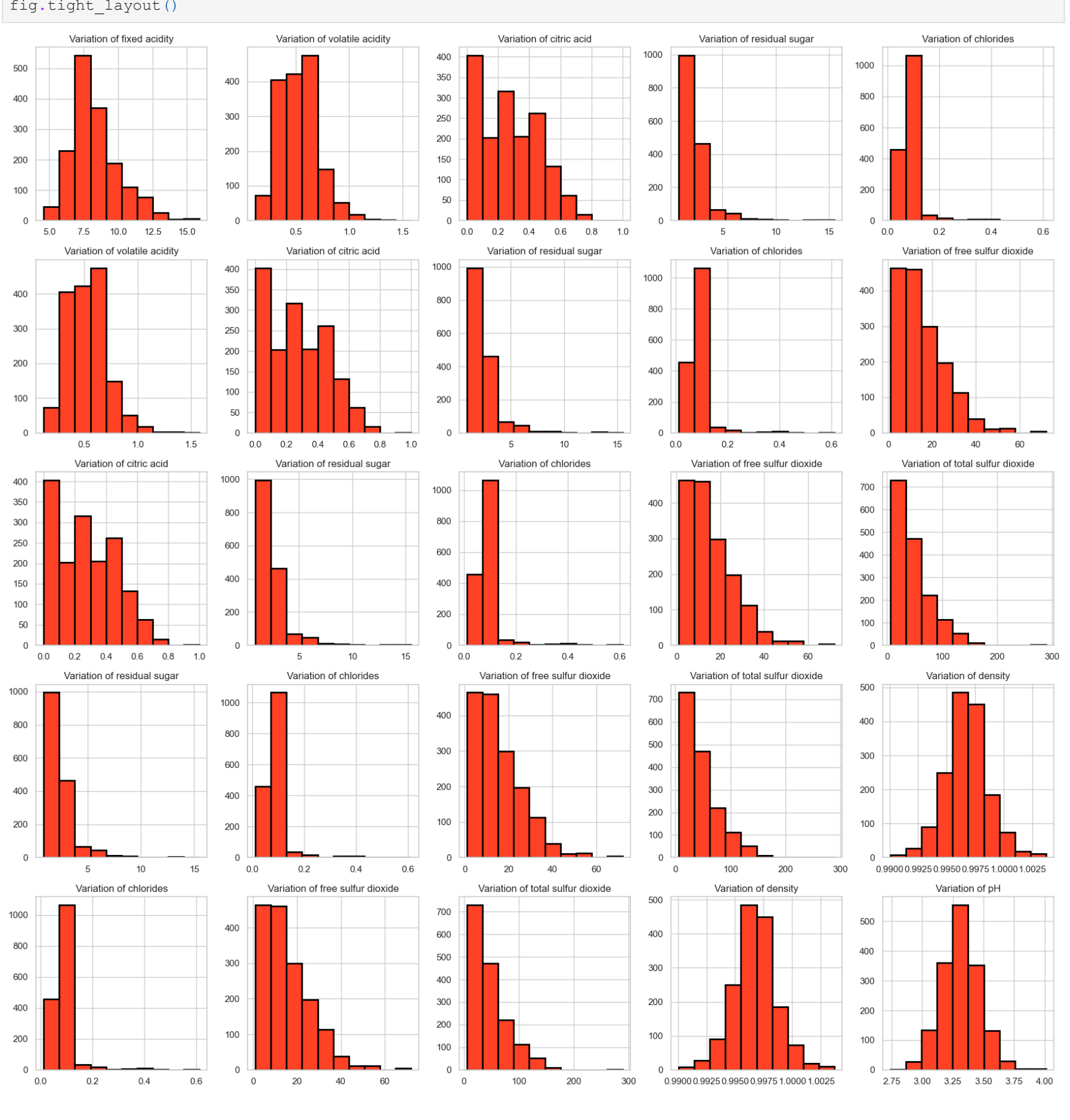
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113	0.658149
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0.169507
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.550000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0.620000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0.730000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.000000

In [9]:
```python
#fixed acidity.
sns.catplot(data=df,kind='box',height=10,aspect=2.5)
```

Out[9]:
```
<seaborn.axisgrid.FacetGrid at 0x27673c5f7d0>
```



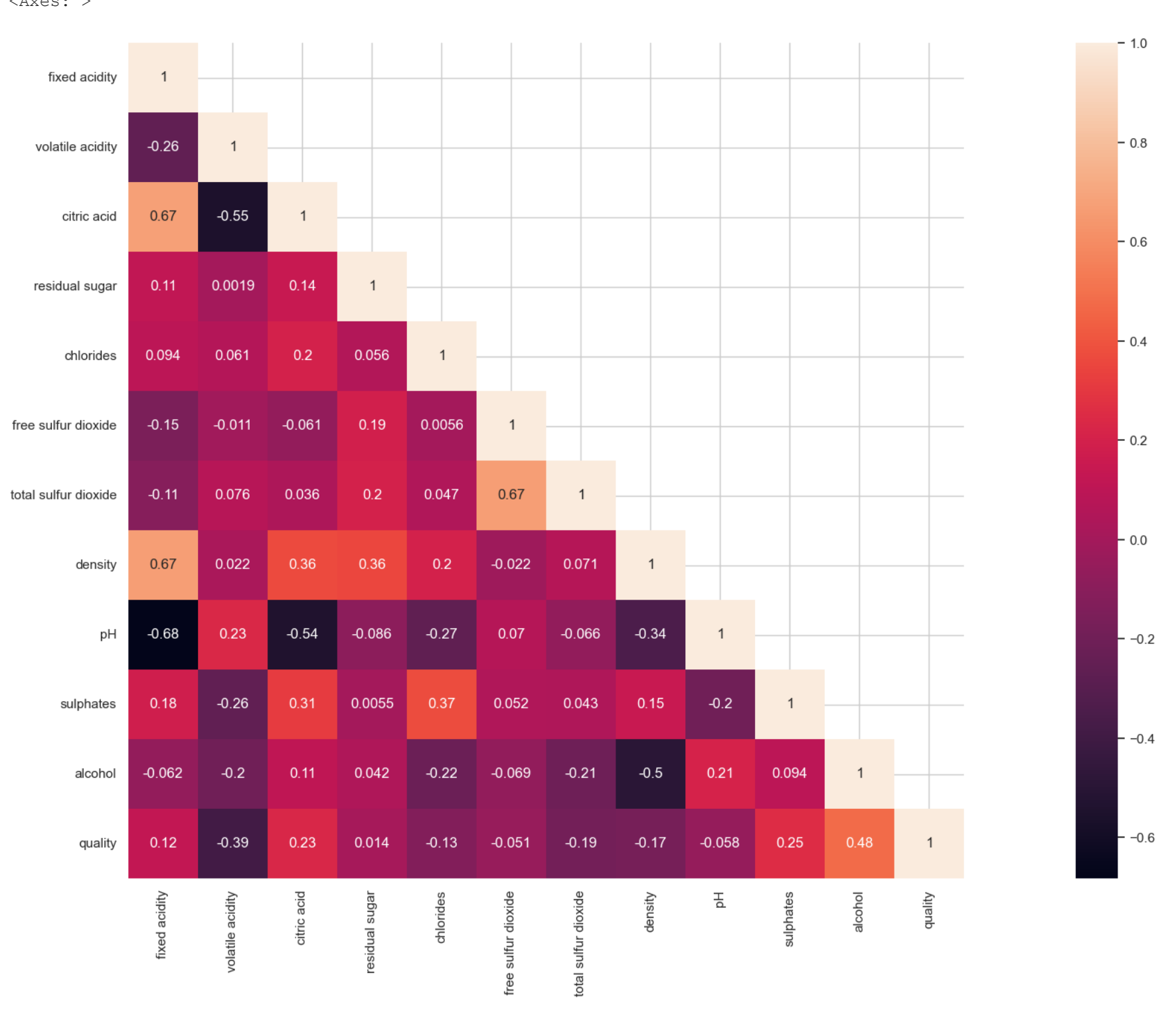
In [10]:
```python
# using a histogram.
fig,axes=plt.subplots(5,5)
columns=['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality']
for i in range (5):
    for j in range (5):
        axes[i,j].hist(x=columns[i+j],data=df,edgecolor='#000000',linewidth=2,color='#ff4125')
        axes[i,j].set_title('Variation of '+columns[i+j])
fig=plt.gcf()
fig.set_size_inches(18,18)
fig.tight_layout()
```



CORELATION BETWEEN DIFFERENT FEATURES

In [11]:
```python
#corelation matrix.
cor_mat= df.corr()
mask = np.array(cor_mat)
mask[np.tril_indices_from(mask)] = False
fig=plt.gcf()
fig.set_size_inches(30,12)
sns.heatmap(data=cor_mat,mask=mask,square=True,annot=True,cbar=True)
```

Out[11]:
```
<Axes: >
```

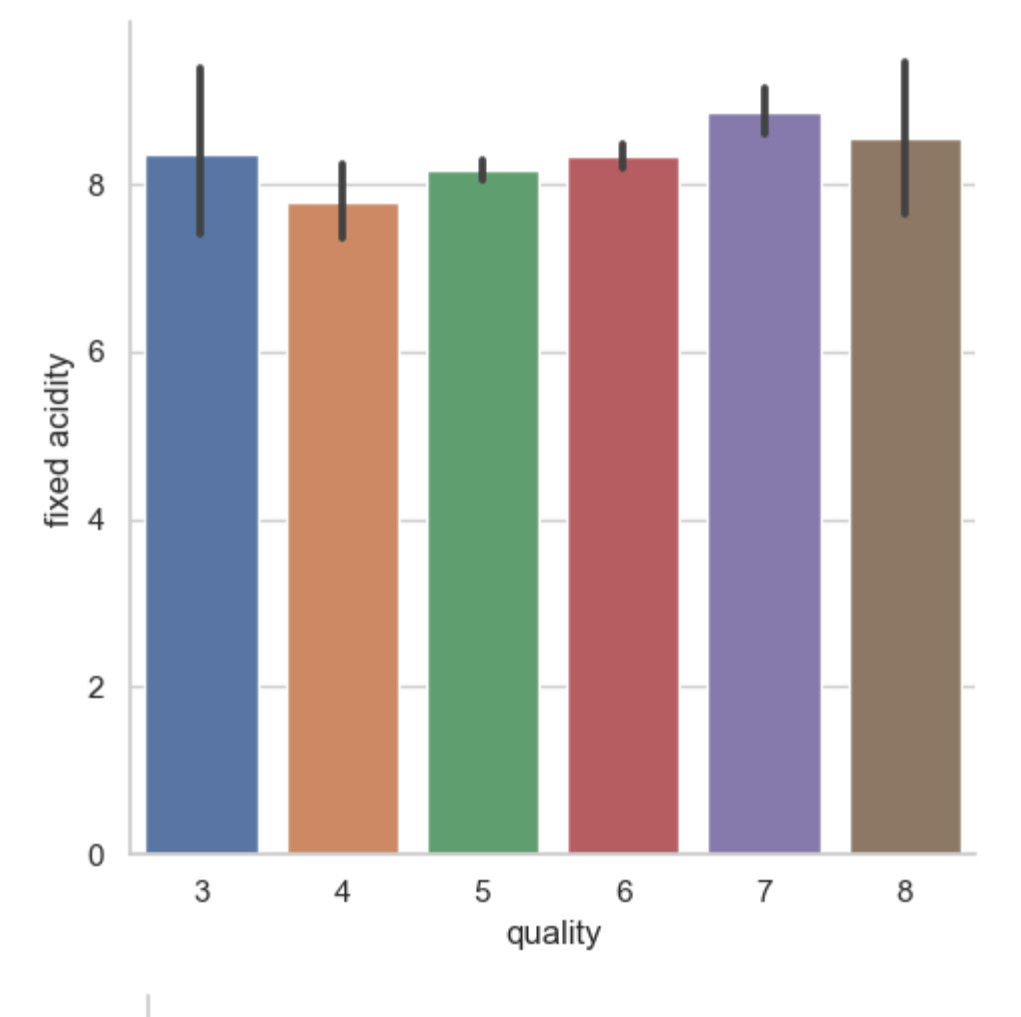


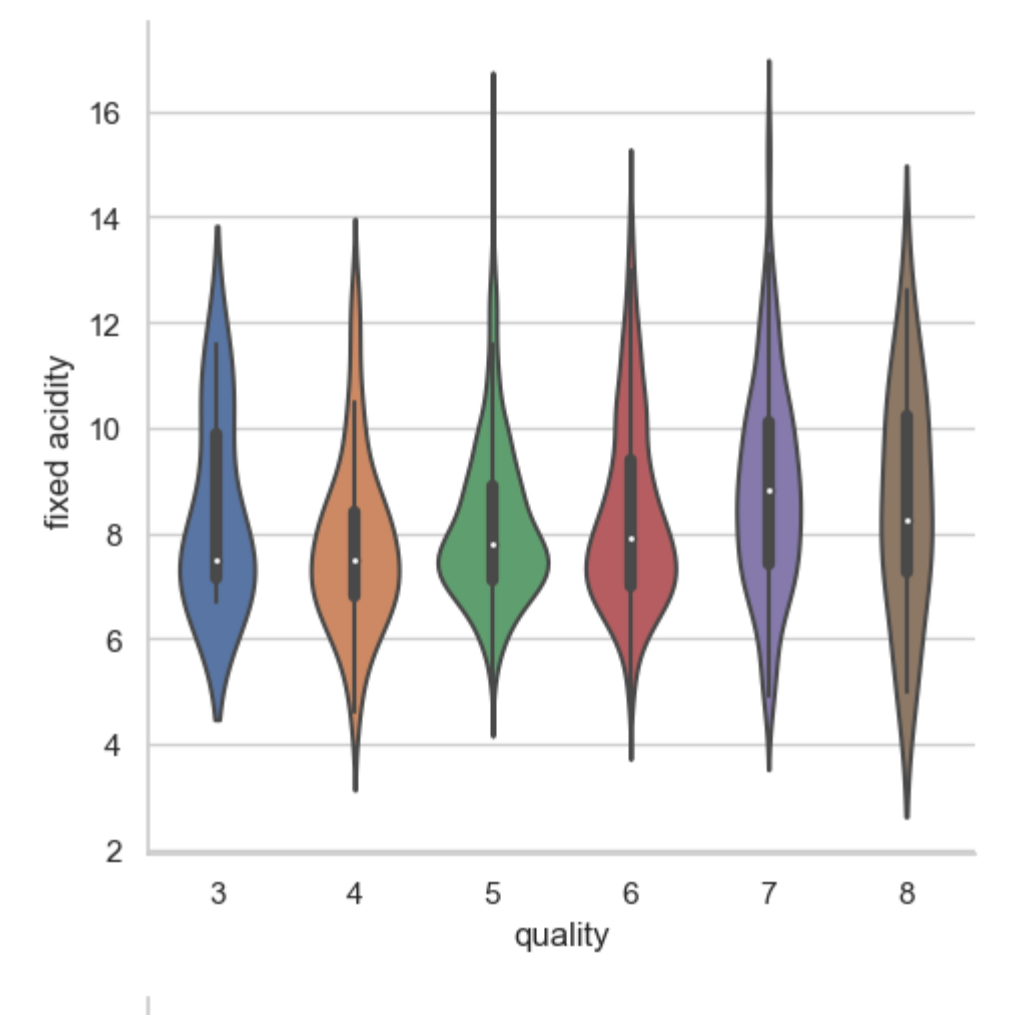
1. The quality of wine is highly related to volatile acidity.
2. Also the quality of wine is highly corelated to alcohol.
3. pH and citric acid/ fixed acidity are highly inversely related as all of us know that acids have smaller pH values.
4. Self Relation ie of a fetaure to itself is 1 as expected.
5. some other similar inferences can be drawn.

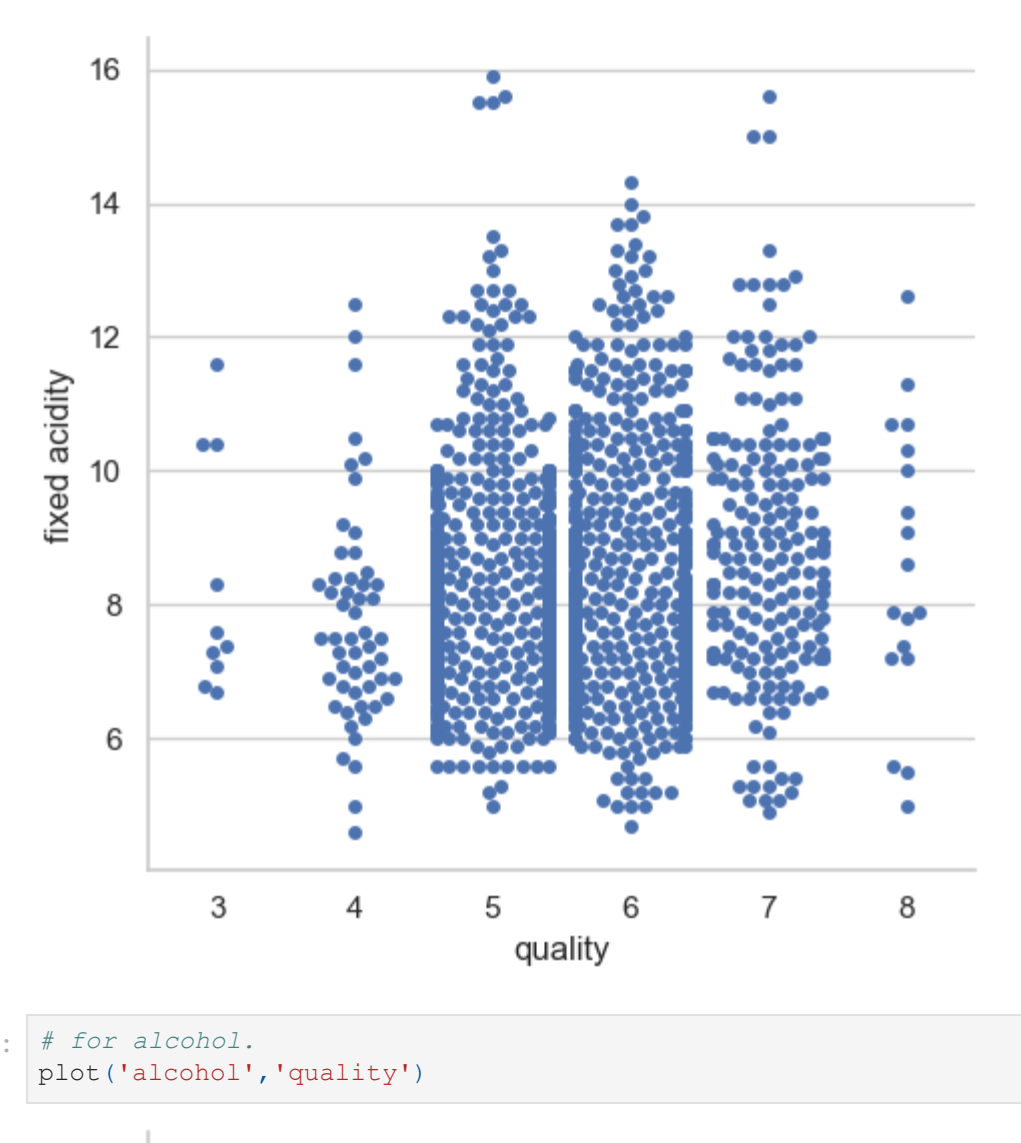
HOW QUALITY VARIES WITH DIFFERENT NUMERIC FEATURES.

In [12]:
```python
def plot(feature_x,target='quality'):
    sns.catplot(x=target,y=feature_x,data=df,kind='bar',height=5,aspect=1)
    sns.catplot(x=target,y=feature_x,data=df,kind='violin',height=5,aspect=1)
    sns.catplot(x=target,y=feature_x,data=df,kind='swarm',height=5,aspect=1)
```

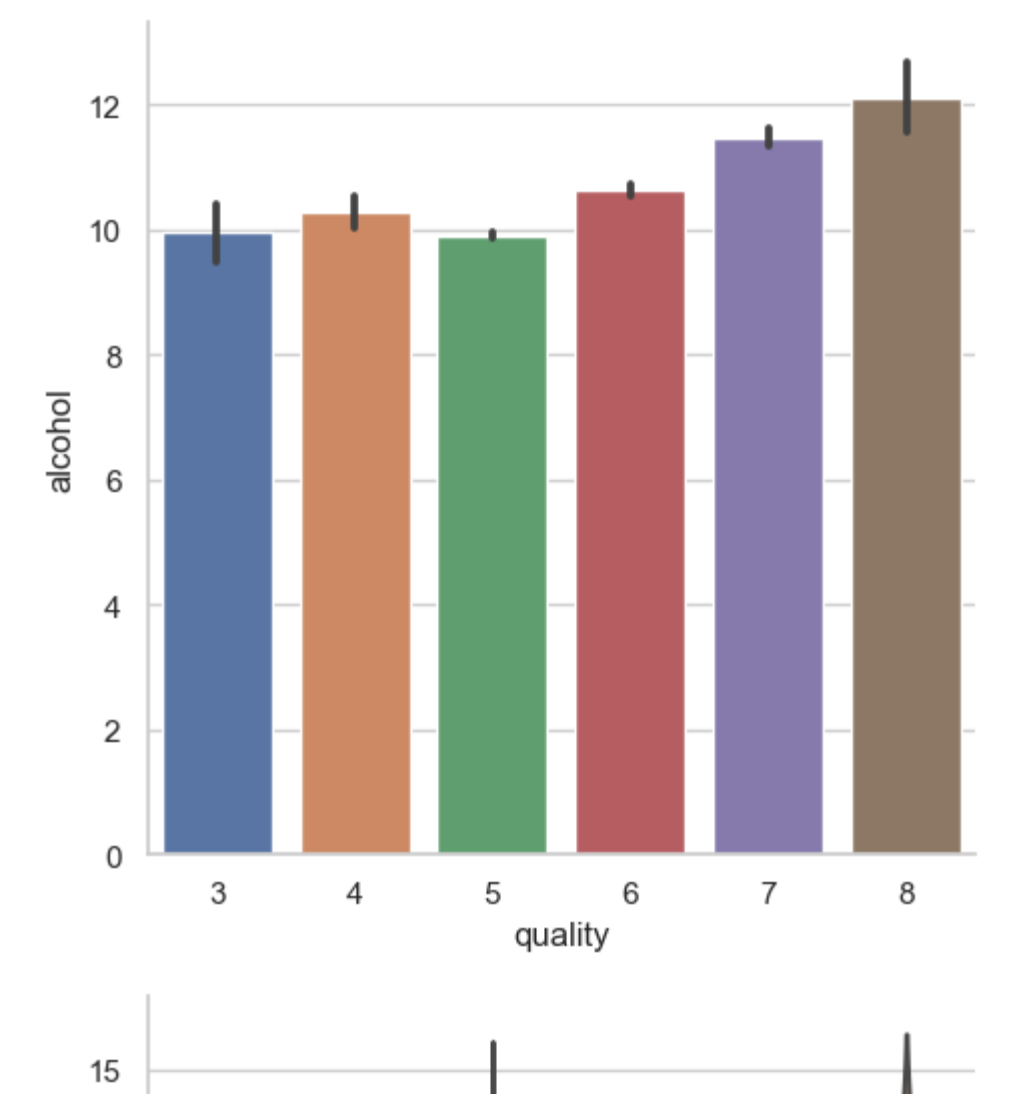
In [13]:
```python
# for fixed acidity.
plot('fixed acidity','quality')
```

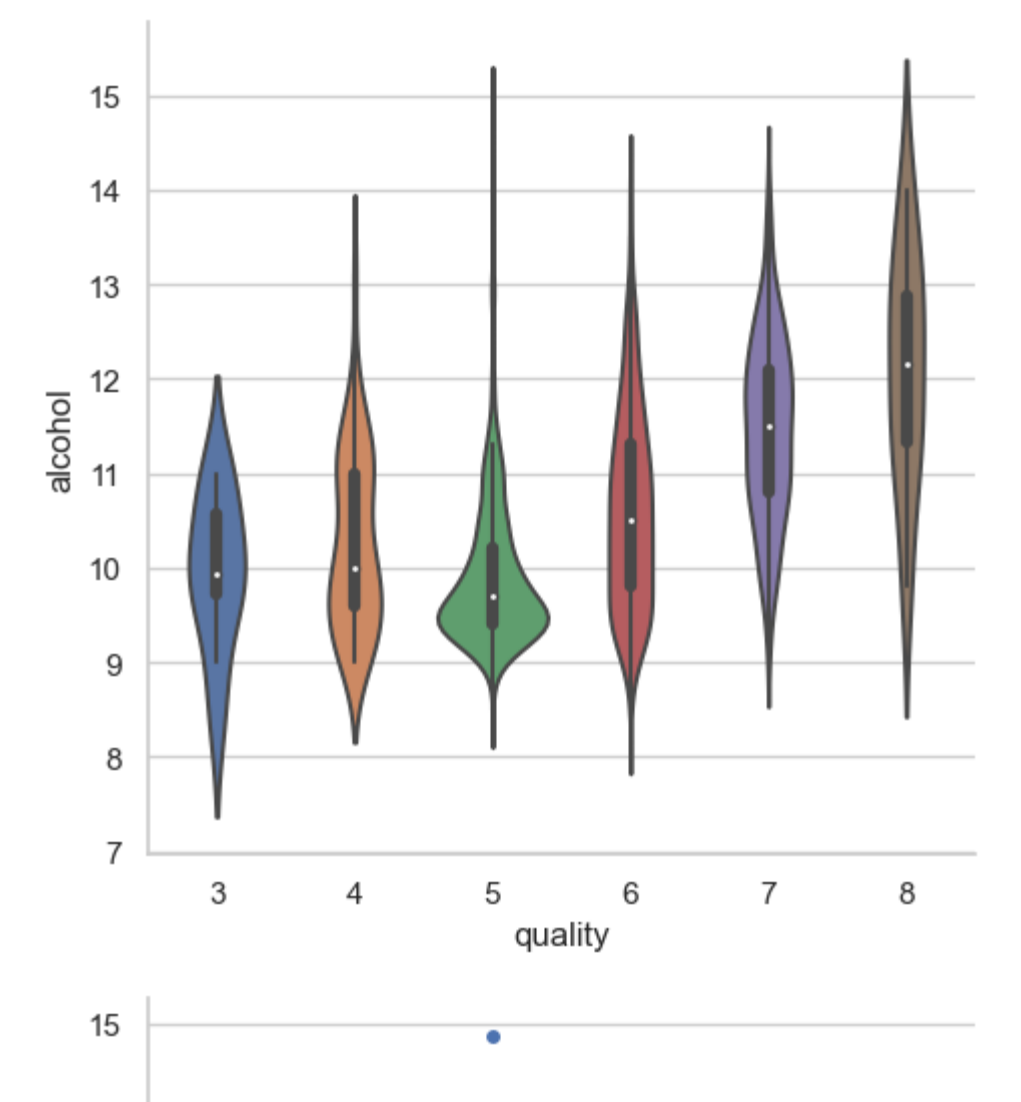


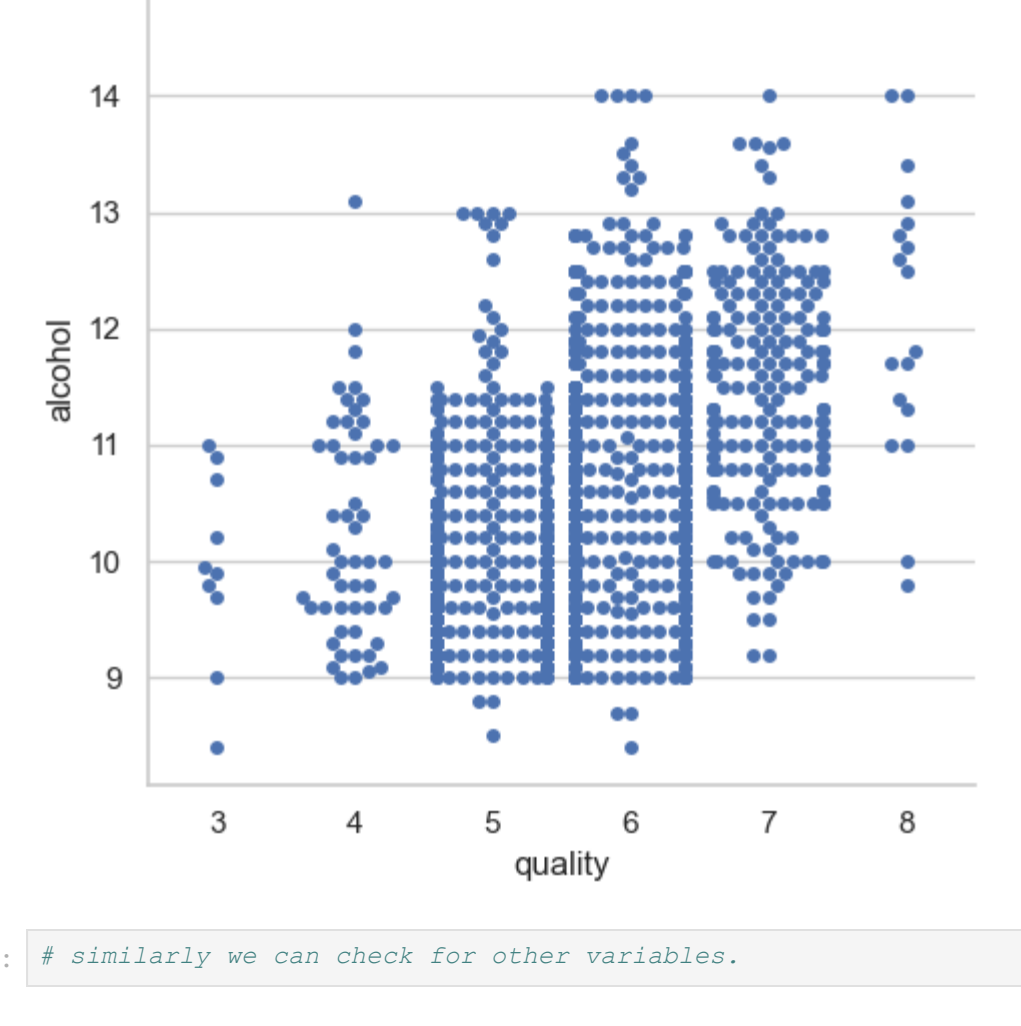




In [14]:
```python
# for alcohol.
plot('alcohol','quality')
```







In [15]:
```python
# similarly we can check for other variables.
```

MODELLING

In [16]:
```python
bins = (2, 6.5, 8)
group_names = ['bad', 'good']
df['quality'] = pd.cut(df['quality'], bins = bins, labels = group_names)
```

In [17]:
```python
label_quality = LabelEncoder()
```

In [18]:
```python
#Bad becomes 0 and good becomes 1
df['quality'] = label_quality.fit_transform(df['quality'])
```

In [19]:
```python
x_train,x_test,y_train,y_test=train_test_split(df.drop('quality',axis=1),df['quality'],test_size=0.25,random_st
```

In [20]:
```python
models=[LogisticRegression(),LinearSVC(),SVC(kernel='rbf'),KNeighborsClassifier(),RandomForestClassifier(),
        DecisionTreeClassifier(),GradientBoostingClassifier(),GaussianNB()]
model_names=['LogisticRegression','LinearSVM','rbfSVM','KNearestNeighbors','RandomForestClassifier','DecisionTr
             'GradientBoostingClassifier','GaussianNB']

acc=[]
d={}

for model in range(len(models)):
    clf=models[model]
    clf.fit(x_train,y_train)
    pred=clf.predict(x_test)
    acc.append(accuracy_score(pred,y_test))

d={'Modelling Algo':model_names,'Accuracy':acc}
d
```

Out[20]:
```
{'Modelling Algo': ['LogisticRegression',
  'LinearSVM',
  'rbfSVM',
  'KNearestNeighbors',
  'RandomForestClassifier',
  'DecisionTree',
  'GradientBoostingClassifier',
  'GaussianNB'],
 'Accuracy': [0.875, 0.875, 0.87, 0.8625, 0.91, 0.895, 0.8775, 0.8525]}
```

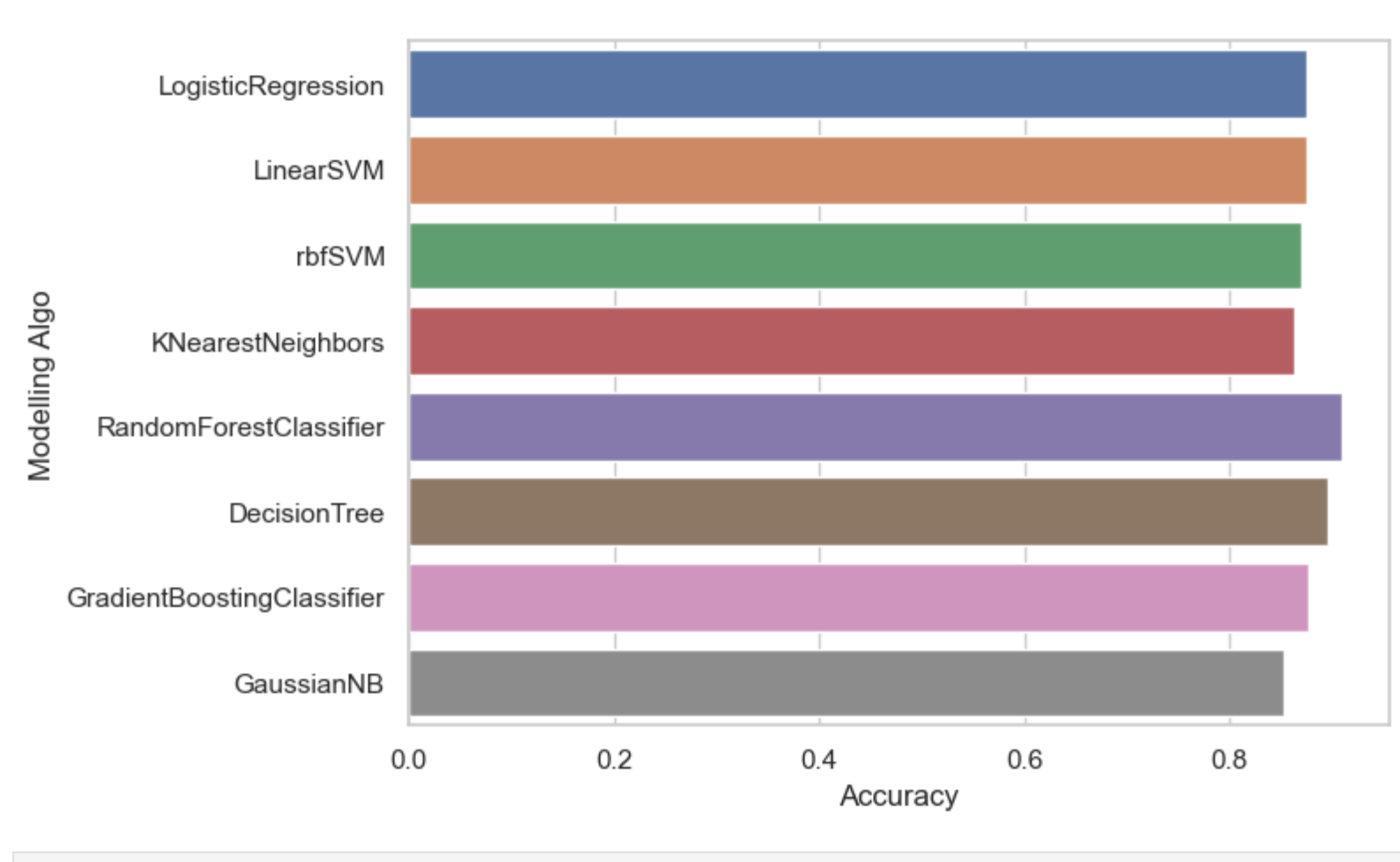
In [21]:
```python
acc_frame=pd.DataFrame(d)
acc_frame
```

Out[21]:

	Modelling Algo	Accuracy
0	LogisticRegression	0.8750
1	LinearSVM	0.8750
2	rbfSVM	0.8700
3	KNearestNeighbors	0.8625
4	RandomForestClassifier	0.9100
5	DecisionTree	0.8950
6	GradientBoostingClassifier	0.8775
7	GaussianNB	0.8525

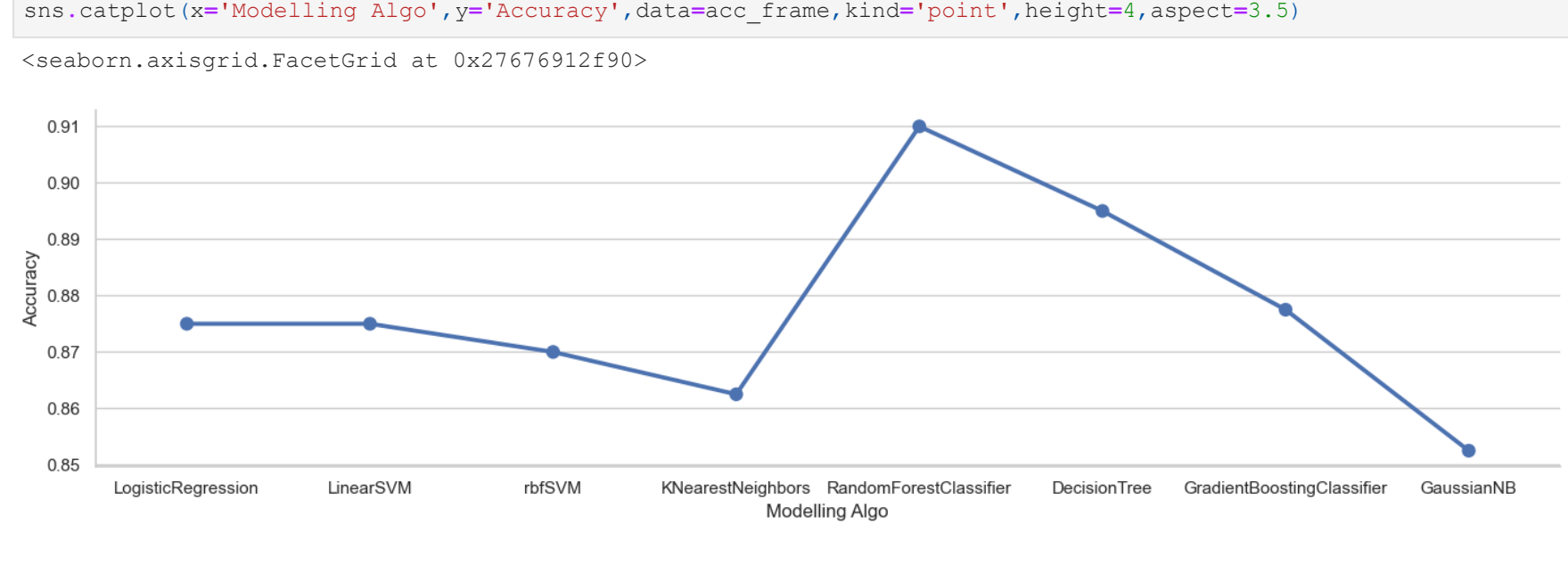
In [22]:
```python
sns.barplot(y='Modelling Algo',x='Accuracy',data=acc_frame)
```

Out[22]:
```
<Axes: xlabel='Accuracy', ylabel='Modelling Algo'>
```



In [23]:
```python
sns.catplot(x='Modelling Algo',y='Accuracy',data=acc_frame,kind='point',height=4,aspect=3.5)
```

Out[23]:
```
<seaborn.axisgrid.FacetGrid at 0x27676912f90>
```



THIS IS WITHOUT FEATURE SCALING. NOW SINCE FEATURES HAVE DIFFERENT SCALES LET US TRY TO DO FEATURE SCALING AND SEE THE IMPACT.

In [24]:
```python
def func(x_train,x_test,y_train,y_test,name_scaler):
    models=[LogisticRegression(),LinearSVC(),SVC(kernel='rbf'),KNeighborsClassifier(),RandomForestClassifier(),
        DecisionTreeClassifier(),GradientBoostingClassifier(),GaussianNB()]
    acc_sc=[]
    for model in range(len(models)):
        clf=models[model]
        clf.fit(x_train,y_train)
        pred=clf.predict(x_test)
        acc_sc.append(accuracy_score(pred,y_test))

    acc_frame[name_scaler]=np.array(acc_sc)
```

In [25]:
```python
scalers=[MinMaxScaler(),StandardScaler()]
names=['Acc_Min_Max_Scaler','Acc_Standard_Scaler']
for scale in range(len(scalers)):
    scaler=scalers[scale]
    scaler.fit(df)
    scaled_df=scaler.transform(df)
    X=scaled_df[:,0:11]
    Y=df['quality'].to_numpy()
    x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.25,random_state=42)
    func(x_train,x_test,y_train,y_test,names[scale])
```

In [26]:
```python
acc_frame
```

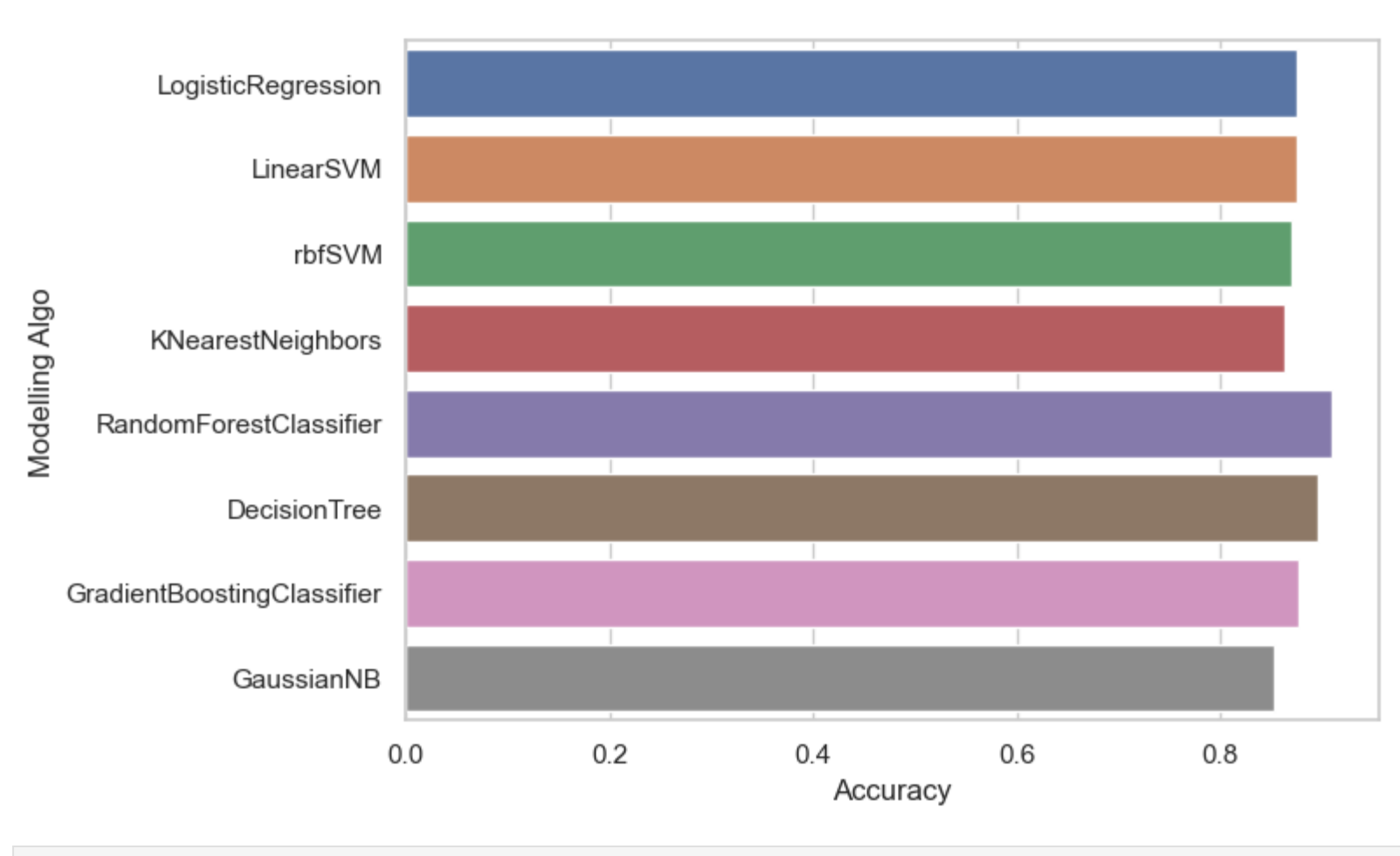
Out[26]:

	Modelling Algo	Accuracy	Acc_Min_Max_Scaler	Acc_Standard_Scaler
0	LogisticRegression	0.8750	0.8800	0.8775
1	LinearSVM	0.8750	0.8825	0.8825
2	rbfSVM	0.8700	0.8850	0.8900
3	KNearestNeighbors	0.8625	0.8850	0.8950
4	RandomForestClassifier	0.9100	0.9100	0.9025
5	DecisionTree	0.8950	0.8925	0.8925
6	GradientBoostingClassifier	0.8775	0.8750	0.8775
7	GaussianNB	0.8525	0.8425	0.8425

1. Note that here the accuracies increase marginally on scaling.
2. Also for this data, StandardScaling seems to give slightly better results than the MinMaxScaling.
3. For some modelling algos there is a considerable increase in accuracies upon scaling the features like SVM, KNN wheras for others there isn't a considerable increase in accuracies upon scaling.

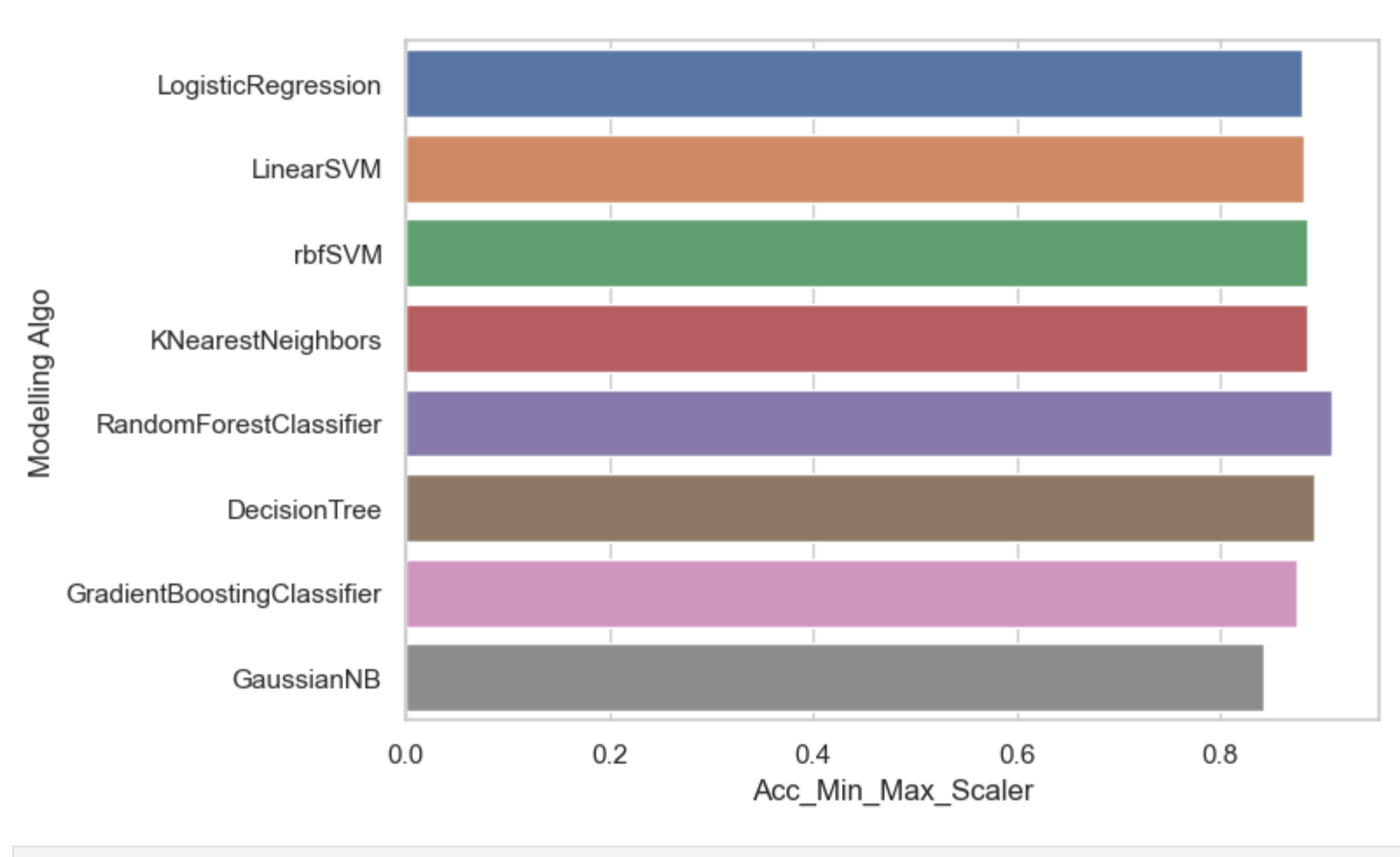
In [27]:
```python
# just to visualize the accuracies.
sns.barplot(y='Modelling Algo',x='Accuracy',data=acc_frame)
```

Out[27]:
```
<Axes: xlabel='Accuracy', ylabel='Modelling Algo'>
```



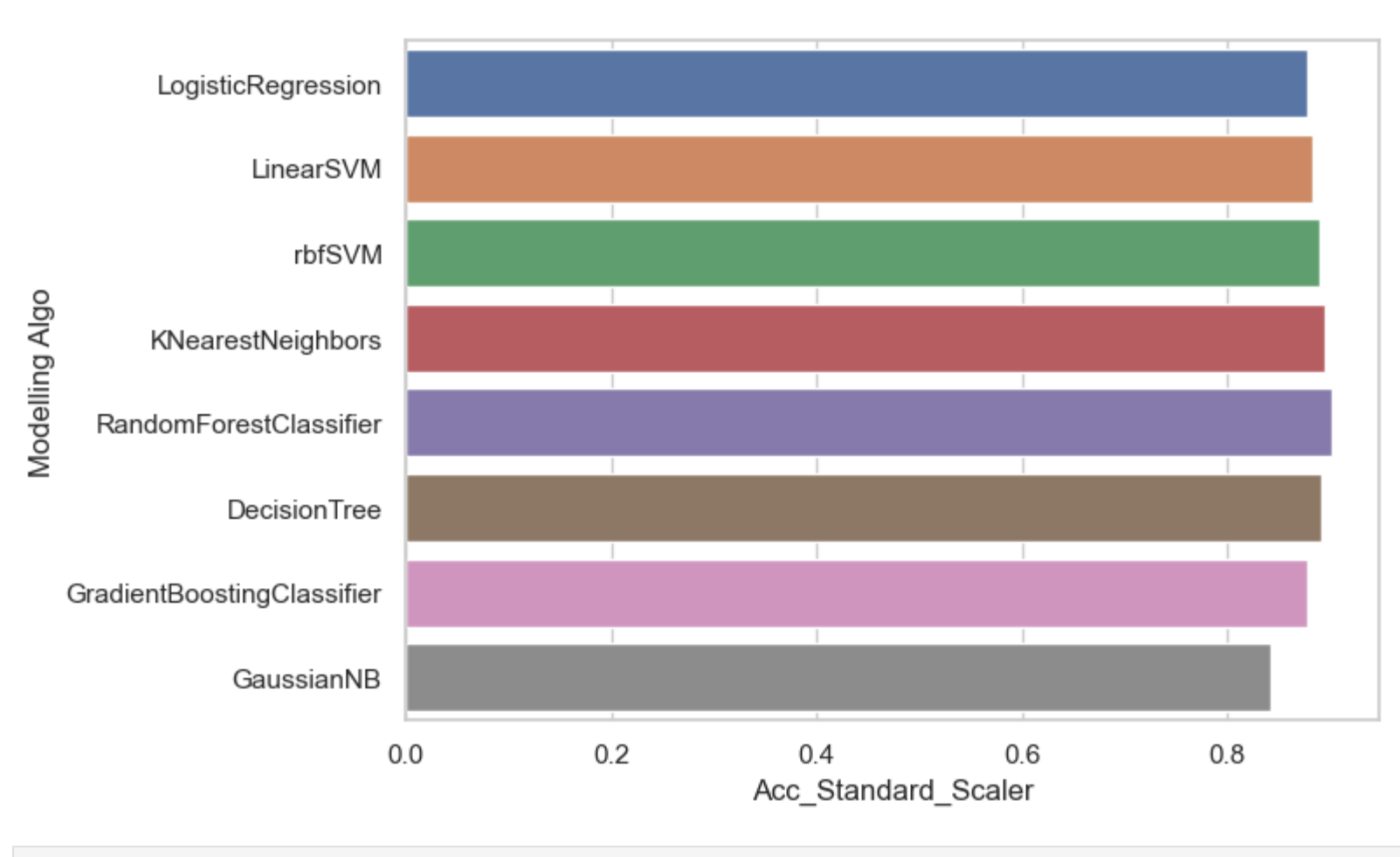
In [28]:
```python
sns.barplot(y='Modelling Algo',x='Acc_Min_Max_Scaler',data=acc_frame)
```

Out[28]:
```
<Axes: xlabel='Acc_Min_Max_Scaler', ylabel='Modelling Algo'>
```



In [29]:
```python
sns.barplot(y='Modelling Algo',x='Acc_Standard_Scaler',data=acc_frame)
```

Out[29]:
```
<Axes: xlabel='Acc_Standard_Scaler', ylabel='Modelling Algo'>
```



In [30]:
```python
# preparing the features by using a StandardScaler as it gave better resluts.
scaler=StandardScaler()
scaled_df=scaler.fit_transform(df)
X=scaled_df[:,0:11]
Y=df['quality'].to_numpy()
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.25,random_state=42)
```

PARAMETER TUNING

1. LOGISTIC REGRESSION.

In [31]:
```python
params_dict={'C':[0.001, 0.01, 0.1, 1, 10, 100, 1000],'penalty':['l1','l2']}
clf_lr=GridSearchCV(estimator=LogisticRegression(),param_grid=params_dict,scoring='accuracy',cv=10)
clf_lr.fit(x_train,y_train)
```

Out[31]:
```
GridSearchCV
estimator: LogisticRegression
LogisticRegression
```

In [32]:
```python
clf_lr.best_params_
```

Out[32]:
```
{'C': 0.1, 'penalty': 'l2'}
```

In [33]:
```python
clf_lr.best_score_ # the best accuracy obtained by Grid search on the train set.
```

Out[33]:
```
0.8815826330532213
```

In [34]:
```python
clf_lr.cv_results_
```

```
Out[34]: {'mean_fit_time': array([0.00059965, 0.00360003, 0.00050023, 0.0028002 , 0.00059991,
        0.00410001, 0.00050004, 0.00499988, 0.0003    , 0.0052002 ,
        0.00040016, 0.00460021, 0.00040004, 0.00540028]),
 'std_fit_time': array([0.00048961, 0.0011136 , 0.00050023, 0.000872  , 0.00048982,
        0.00069991, 0.00050004, 0.00100031, 0.00045826, 0.00087161,
        0.0004901 , 0.00066332, 0.00048995, 0.00049019]),
 'mean_score_time': array([0.        , 0.00120029, 0.        , 0.00089977, 0.        ,
        0.00090034, 0.        , 0.00110044, 0.        , 0.00120037,
        0.        , 0.00090017, 0.        , 0.00110018]),
 'std_score_time': array([0.        , 0.00040002, 0.        , 0.00029992, 0.        ,
        0.00030011, 0.        , 0.00053851, 0.        , 0.00040016,
        0.        , 0.00030006, 0.        , 0.00029997]),
 'param_C': masked_array(data=[0.001, 0.001, 0.01, 0.01, 0.1, 0.1, 1, 1, 10, 10, 100,
                    100, 1000, 1000],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'param_penalty': masked_array(data=['l1', 'l2', 'l1', 'l2', 'l1', 'l2', 'l1', 'l2', 'l1',
                    'l2', 'l1', 'l2', 'l1', 'l2'],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'C': 0.001, 'penalty': 'l1'},
  {'C': 0.001, 'penalty': 'l2'},
  {'C': 0.01, 'penalty': 'l1'},
  {'C': 0.01, 'penalty': 'l2'},
  {'C': 0.1, 'penalty': 'l1'},
  {'C': 0.1, 'penalty': 'l2'},
  {'C': 1, 'penalty': 'l1'},
  {'C': 1, 'penalty': 'l2'},
  {'C': 10, 'penalty': 'l1'},
  {'C': 10, 'penalty': 'l2'},
  {'C': 100, 'penalty': 'l1'},
  {'C': 100, 'penalty': 'l2'},
  {'C': 1000, 'penalty': 'l1'},
  {'C': 1000, 'penalty': 'l2'}],
 'split0_test_score': array([       nan, 0.86666667,        nan, 0.875     ,        nan,
        0.86666667,        nan, 0.85833333,        nan, 0.85833333,
               nan, 0.85      ,        nan, 0.85      ]),
 'split1_test_score': array([       nan, 0.86666667,        nan, 0.875     ,        nan,
        0.88333333,        nan, 0.88333333,        nan, 0.89166667,
               nan, 0.89166667,        nan, 0.89166667]),
 'split2_test_score': array([       nan, 0.86666667,        nan, 0.86666667,        nan,
        0.875     ,        nan, 0.88333333,        nan, 0.88333333,
               nan, 0.88333333,        nan, 0.88333333]),
 'split3_test_score': array([       nan, 0.86666667,        nan, 0.86666667,        nan,
        0.91666667,        nan, 0.90833333,        nan, 0.90833333,
               nan, 0.90833333,        nan, 0.90833333]),
 'split4_test_score': array([       nan, 0.86666667,        nan, 0.86666667,        nan,
        0.88333333,        nan, 0.875     ,        nan, 0.875     ,
               nan, 0.875     ,        nan, 0.875     ]),
 'split5_test_score': array([       nan, 0.85833333,        nan, 0.85      ,        nan,
        0.825     ,        nan, 0.84166667,        nan, 0.84166667,
               nan, 0.84166667,        nan, 0.84166667]),
 'split6_test_score': array([       nan, 0.85833333,        nan, 0.86666667,        nan,
        0.89166667,        nan, 0.88333333,        nan, 0.875     ,
               nan, 0.86666667,        nan, 0.86666667]),
 'split7_test_score': array([       nan, 0.85833333,        nan, 0.875     ,        nan,
        0.90833333,        nan, 0.89166667,        nan, 0.89166667,
               nan, 0.89166667,        nan, 0.89166667]),
 'split8_test_score': array([       nan, 0.85833333,        nan, 0.84166667,        nan,
        0.86666667,        nan, 0.85833333,        nan, 0.85833333,
               nan, 0.85833333,        nan, 0.85833333]),
 'split9_test_score': array([       nan, 0.86554622,        nan, 0.87394958,        nan,
        0.89915966,        nan, 0.89915966,        nan, 0.89915966,
               nan, 0.89915966,        nan, 0.89915966]),
 'mean_test_score': array([       nan, 0.86322129,        nan, 0.86572829,        nan,
        0.88158263,        nan, 0.8782493 ,        nan, 0.8782493 ,
               nan, 0.87658263,        nan, 0.87658263]),
 'std_test_score': array([       nan, 0.00400408,        nan, 0.0107487 ,        nan,
        0.02460316,        nan, 0.01934425,        nan, 0.01969997,
               nan, 0.02092388,        nan, 0.02092388]),
 'rank_test_score': array([8, 7, 8, 6, 8, 1, 8, 2, 8, 2, 8, 4, 8, 4])}
```

```
In [35]: pred=clf_lr.predict(x_test)
         accuracy_score(pred,y_test)
```

```
Out[35]: 0.8825
```

2. KNN.

```
In [36]: l=[i+1 for i in range(50)]
         params_dict={'n_neighbors':l,'n_jobs':[-1]}
         clf_knn=GridSearchCV(estimator=KNeighborsClassifier(),param_grid=params_dict,scoring='accuracy',cv=10)
         clf_knn.fit(x_train,y_train)
```

```
Out[36]: ▸ GridSearchCV
         ▸ estimator: KNeighborsClassifier
             ▸ KNeighborsClassifier
```

```
In [37]: clf_knn.best_score_
```

```
Out[37]: 0.8807422969187675
```

```
In [38]: clf_knn.best_params_
```

```
Out[38]: {'n_jobs': -1, 'n_neighbors': 1}
```

```
In [39]: clf_knn.cv_results_
```

```
Out[39]: {'mean_fit_time': array([0.00360007, 0.00329976, 0.00349979, 0.00329914, 0.00359979,
        0.00319986, 0.00340011, 0.00330005, 0.00349929, 0.0033998 ,
        0.00309944, 0.00329969, 0.00309961, 0.00339971, 0.00340002,
        0.00330007, 0.00339997, 0.00339983, 0.00339932, 0.00349951,
        0.00339944, 0.00359986, 0.00349982, 0.0030998 , 0.00349998,
        0.00329995, 0.00349941, 0.00309978, 0.00339997, 0.00359991,
        0.00450037, 0.00349987, 0.00369992, 0.00329986, 0.00359962,
        0.00349989, 0.00359962, 0.0033721 , 0.00369949, 0.00339975,
        0.0033    , 0.00339973, 0.00359983, 0.00329936, 0.00330002,
        0.00329981, 0.00349987, 0.00349989, 0.00339966, 0.0033999 ]),
 'std_fit_time': array([0.00066395, 0.00045854, 0.00049992, 0.00064052, 0.00180049,
        0.0004001 , 0.00048991, 0.00064069, 0.00067075, 0.0006635 ,
        0.00030001, 0.00045791, 0.00030027, 0.00049   , 0.00049023,
        0.00045839, 0.00066333, 0.00049019, 0.00048973, 0.00050049,
        0.00048982, 0.0004902 , 0.00050008, 0.00029997, 0.00049944,
        0.00045872, 0.00049996, 0.00030022, 0.00048959, 0.00101996,
        0.00210993, 0.00092257, 0.0010054 , 0.00045837, 0.00091674,
        0.00050011, 0.00048996, 0.00046239, 0.00045807, 0.00048923,
        0.00045802, 0.00066336, 0.00048931, 0.00045823, 0.00045847,
        0.00064002, 0.0008063 , 0.00067095, 0.00049014, 0.00049043]),
 'mean_score_time': array([0.0217865 , 0.02050881, 0.02054732, 0.02043056, 0.02150202,
        0.02062187, 0.02069428, 0.02048807, 0.02119935, 0.02060492,
        0.02030835, 0.02073259, 0.02064486, 0.02057819, 0.02054389,
        0.02057512, 0.02031496, 0.02207763, 0.02086515, 0.02080796,
        0.02048311, 0.02070975, 0.02070394, 0.02045312, 0.02113431,
        0.02033901, 0.02092433, 0.02080922, 0.0236783 , 0.02061551,
        0.02253811, 0.02029135, 0.02381692, 0.02168586, 0.02192054,
        0.02078629, 0.02090485, 0.01980689, 0.02068667, 0.0263968 ,
        0.02013795, 0.02050462, 0.02362342, 0.02038581, 0.02082694,
        0.020609  , 0.02398546, 0.02086794, 0.02054255, 0.02028043]),
 'std_score_time': array([0.00375416, 0.00072331, 0.00051714, 0.00038057, 0.00162822,
        0.0009379 , 0.00044186, 0.00103061, 0.00115504, 0.00099594,
        0.00062315, 0.00059503, 0.00051714, 0.0006916 , 0.00043607,
        0.00058466, 0.00059798, 0.00294656, 0.00048212, 0.00062069,
        0.00069954, 0.0005657 , 0.00047096, 0.00068463, 0.00077446,
        0.00064352, 0.00065301, 0.00079222, 0.00597825, 0.00134099,
        0.00216491, 0.00155124, 0.00286163, 0.00264898, 0.00461298,
        0.00052709, 0.0008658 , 0.00053802, 0.00082221, 0.01450194,
        0.0007806 , 0.00071295, 0.00825634, 0.00047378, 0.00085897,
        0.00085628, 0.01069798, 0.00068406, 0.000739  , 0.00061249]),
 'param_n_jobs': masked_array(data=[-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
                    -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
                    -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
                    -1, -1, -1, -1, -1, -1, -1, -1],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'param_n_neighbors': masked_array(data=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
                    17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
                    31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44,
                    45, 46, 47, 48, 49, 50],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False],
        fill_value='?',
             dtype=object),
 'params': [{'n_jobs': -1, 'n_neighbors': 1},
  {'n_jobs': -1, 'n_neighbors': 2},
  {'n_jobs': -1, 'n_neighbors': 3},
  {'n_jobs': -1, 'n_neighbors': 4},
  {'n_jobs': -1, 'n_neighbors': 5},
  {'n_jobs': -1, 'n_neighbors': 6},
  {'n_jobs': -1, 'n_neighbors': 7},
  {'n_jobs': -1, 'n_neighbors': 8},
  {'n_jobs': -1, 'n_neighbors': 9},
  {'n_jobs': -1, 'n_neighbors': 10},
  {'n_jobs': -1, 'n_neighbors': 11},
  {'n_jobs': -1, 'n_neighbors': 12},
  {'n_jobs': -1, 'n_neighbors': 13},
  {'n_jobs': -1, 'n_neighbors': 14},
  {'n_jobs': -1, 'n_neighbors': 15},
  {'n_jobs': -1, 'n_neighbors': 16},
  {'n_jobs': -1, 'n_neighbors': 17},
  {'n_jobs': -1, 'n_neighbors': 18},
  {'n_jobs': -1, 'n_neighbors': 19},
  {'n_jobs': -1, 'n_neighbors': 20},
  {'n_jobs': -1, 'n_neighbors': 21},
  {'n_jobs': -1, 'n_neighbors': 22},
  {'n_jobs': -1, 'n_neighbors': 23},
  {'n_jobs': -1, 'n_neighbors': 24},
  {'n_jobs': -1, 'n_neighbors': 25},
  {'n_jobs': -1, 'n_neighbors': 26},
  {'n_jobs': -1, 'n_neighbors': 27},
  {'n_jobs': -1, 'n_neighbors': 28},
  {'n_jobs': -1, 'n_neighbors': 29},
  {'n_jobs': -1, 'n_neighbors': 30},
  {'n_jobs': -1, 'n_neighbors': 31},
  {'n_jobs': -1, 'n_neighbors': 32},
  {'n_jobs': -1, 'n_neighbors': 33},
  {'n_jobs': -1, 'n_neighbors': 34},
  {'n_jobs': -1, 'n_neighbors': 35},
  {'n_jobs': -1, 'n_neighbors': 36},
  {'n_jobs': -1, 'n_neighbors': 37},
  {'n_jobs': -1, 'n_neighbors': 38},
  {'n_jobs': -1, 'n_neighbors': 39},
  {'n_jobs': -1, 'n_neighbors': 40},
  {'n_jobs': -1, 'n_neighbors': 41},
  {'n_jobs': -1, 'n_neighbors': 42},
  {'n_jobs': -1, 'n_neighbors': 43},
  {'n_jobs': -1, 'n_neighbors': 44},
  {'n_jobs': -1, 'n_neighbors': 45},
  {'n_jobs': -1, 'n_neighbors': 46},
  {'n_jobs': -1, 'n_neighbors': 47},
  {'n_jobs': -1, 'n_neighbors': 48},
  {'n_jobs': -1, 'n_neighbors': 49},
  {'n_jobs': -1, 'n_neighbors': 50}],
 'split0_test_score': array([0.875     , 0.85      , 0.85833333, 0.85      , 0.83333333,
        0.84166667, 0.84166667, 0.84166667, 0.83333333, 0.84166667,
        0.84166667, 0.84166667, 0.84166667, 0.84166667, 0.84166667,
        0.84166667, 0.84166667, 0.84166667, 0.84166667, 0.83333333,
        0.84166667, 0.84166667, 0.83333333, 0.84166667, 0.83333333,
        0.84166667, 0.84166667, 0.84166667, 0.84166667, 0.84166667,
        0.84166667, 0.85      , 0.83333333, 0.85      , 0.85      ,
        0.84166667, 0.85      , 0.84166667, 0.84166667, 0.84166667,
        0.84166667, 0.84166667, 0.84166667, 0.84166667, 0.84166667,
        0.84166667, 0.84166667, 0.85      , 0.85      , 0.85      ]),
 'split1_test_score': array([0.90833333, 0.9       , 0.9       , 0.89166667, 0.89166667,
        0.89166667, 0.89166667, 0.90833333, 0.89166667, 0.9       ,
        0.91666667, 0.90833333, 0.90833333, 0.9       , 0.9       ,
        0.90833333, 0.9       , 0.90833333, 0.91666667, 0.91666667,
        0.91666667, 0.90833333, 0.90833333, 0.9       , 0.90833333,
        0.90833333, 0.90833333, 0.9       , 0.9       , 0.9       ,
        0.90833333, 0.89166667, 0.89166667, 0.89166667, 0.89166667,
        0.89166667, 0.89166667, 0.89166667, 0.89166667, 0.89166667,
        0.89166667, 0.89166667, 0.89166667, 0.89166667, 0.89166667,
        0.89166667, 0.89166667, 0.89166667, 0.89166667, 0.89166667]),
 'split2_test_score': array([0.86666667, 0.88333333, 0.875     , 0.88333333, 0.88333333,
        0.86666667, 0.875     , 0.86666667, 0.875     , 0.86666667,
        0.875     , 0.88333333, 0.875     , 0.875     , 0.86666667,
        0.85833333, 0.85833333, 0.85833333, 0.85833333, 0.875     ,
        0.875     , 0.88333333, 0.875     , 0.88333333, 0.875     ,
        0.875     , 0.875     , 0.875     , 0.875     , 0.875     ,
        0.875     , 0.875     , 0.875     , 0.875     , 0.875     ,
        0.875     , 0.875     , 0.875     , 0.875     , 0.88333333,
        0.88333333, 0.88333333, 0.88333333, 0.88333333, 0.88333333,
        0.88333333, 0.88333333, 0.875     , 0.875     , 0.88333333]),
 'split3_test_score': array([0.88333333, 0.9       , 0.89166667, 0.9       , 0.89166667,
        0.89166667, 0.86666667, 0.875     , 0.875     , 0.89166667,
        0.89166667, 0.89166667, 0.88333333, 0.875     , 0.88333333,
        0.875     , 0.88333333, 0.88333333, 0.9       , 0.88333333,
        0.89166667, 0.88333333, 0.89166667, 0.88333333, 0.88333333,
        0.875     , 0.89166667, 0.89166667, 0.89166667, 0.89166667,
        0.9       , 0.89166667, 0.90833333, 0.9       , 0.90833333,
        0.89166667, 0.9       , 0.89166667, 0.89166667, 0.89166667,
        0.9       , 0.89166667, 0.89166667, 0.89166667, 0.88333333,
        0.89166667, 0.89166667, 0.89166667, 0.88333333, 0.88333333]),
 'split4_test_score': array([0.88333333, 0.86666667, 0.88333333, 0.86666667, 0.86666667,
        0.875     , 0.88333333, 0.88333333, 0.86666667, 0.875     ,
        0.875     , 0.88333333, 0.875     , 0.88333333, 0.88333333,
        0.88333333, 0.88333333, 0.875     , 0.86666667, 0.875     ,
        0.875     , 0.88333333, 0.88333333, 0.875     , 0.86666667,
        0.88333333, 0.86666667, 0.86666667, 0.86666667, 0.88333333,
        0.86666667, 0.875     , 0.875     , 0.875     , 0.875     ,
        0.875     , 0.875     , 0.875     , 0.875     , 0.88333333,
        0.875     , 0.875     , 0.875     , 0.88333333, 0.88333333,
        0.88333333, 0.88333333, 0.88333333, 0.875     , 0.88333333]),
 'split5_test_score': array([0.83333333, 0.84166667, 0.80833333, 0.84166667, 0.85833333,
        0.84166667, 0.85      , 0.825     , 0.83333333, 0.83333333,
        0.83333333, 0.83333333, 0.84166667, 0.84166667, 0.84166667,
        0.85833333, 0.84166667, 0.85      , 0.83333333, 0.84166667,
        0.83333333, 0.84166667, 0.84166667, 0.84166667, 0.83333333,
        0.83333333, 0.83333333, 0.83333333, 0.83333333, 0.84166667,
        0.84166667, 0.84166667, 0.84166667, 0.85      , 0.84166667,
        0.84166667, 0.84166667, 0.85      , 0.84166667, 0.85833333,
        0.85833333, 0.86666667, 0.86666667, 0.86666667, 0.85833333,
        0.86666667, 0.86666667, 0.86666667, 0.86666667, 0.86666667]),
 'split6_test_score': array([0.86666667, 0.85833333, 0.85833333, 0.88333333, 0.85      ,
        0.86666667, 0.86666667, 0.88333333, 0.85833333, 0.875     ,
        0.85      , 0.85833333, 0.85833333, 0.85833333, 0.85833333,
        0.88333333, 0.89166667, 0.89166667, 0.88333333, 0.88333333,
        0.88333333, 0.89166667, 0.89166667, 0.89166667, 0.89166667,
        0.89166667, 0.88333333, 0.88333333, 0.89166667, 0.88333333,
        0.89166667, 0.88333333, 0.9       , 0.88333333, 0.88333333,
        0.89166667, 0.89166667, 0.89166667, 0.88333333, 0.88333333,
        0.88333333, 0.88333333, 0.89166667, 0.88333333, 0.88333333,
        0.89166667, 0.89166667, 0.88333333, 0.88333333, 0.89166667]),
 'split7_test_score': array([0.89166667, 0.88333333, 0.86666667, 0.89166667, 0.86666667,
        0.85833333, 0.85      , 0.85833333, 0.85833333, 0.86666667,
        0.85833333, 0.86666667, 0.86666667, 0.875     , 0.875     ,
        0.875     , 0.86666667, 0.86666667, 0.86666667, 0.86666667,
        0.86666667, 0.875     , 0.86666667, 0.85833333, 0.85833333,
        0.85833333, 0.86666667, 0.86666667, 0.86666667, 0.875     ,
        0.875     , 0.875     , 0.875     , 0.875     , 0.875     ,
        0.875     , 0.875     , 0.875     , 0.875     , 0.875     ,
        0.875     , 0.875     , 0.875     , 0.875     , 0.875     ,
        0.88333333, 0.88333333, 0.88333333, 0.88333333, 0.88333333]),
 'split8_test_score': array([0.90833333, 0.89166667, 0.85      , 0.86666667, 0.84166667,
        0.85      , 0.825     , 0.84166667, 0.84166667, 0.85833333,
        0.84166667, 0.84166667, 0.85      , 0.85      , 0.85833333,
        0.85      , 0.85833333, 0.85833333, 0.86666667, 0.85      ,
        0.85833333, 0.85      , 0.86666667, 0.86666667, 0.875     ,
        0.88333333, 0.875     , 0.875     , 0.86666667, 0.86666667,
        0.85833333, 0.85      , 0.85833333, 0.86666667, 0.875     ,
        0.86666667, 0.86666667, 0.85      , 0.86666667, 0.86666667,
        0.86666667, 0.85833333, 0.86666667, 0.85833333, 0.85833333,
        0.85833333, 0.85833333, 0.85833333, 0.85833333, 0.85833333]),
 'split9_test_score': array([0.8907563 , 0.88235294, 0.90756303, 0.92436975, 0.89915966,
        0.88235294, 0.8907563 , 0.8907563 , 0.88235294, 0.8907563 ,
        0.88235294, 0.88235294, 0.87394958, 0.87394958, 0.87394958,
        0.88235294, 0.88235294, 0.88235294, 0.8907563 , 0.8907563 ,
        0.8907563 , 0.8907563 , 0.8907563 , 0.89915966, 0.89915966,
        0.89915966, 0.8907563 , 0.89915966, 0.89915966, 0.89915966,
        0.88235294, 0.89915966, 0.89915966, 0.8907563 , 0.89915966,
        0.89915966, 0.89915966, 0.89915966, 0.89915966, 0.89915966,
        0.89915966, 0.89915966, 0.89915966, 0.90756303, 0.89915966,
        0.89915966, 0.89915966, 0.89915966, 0.89915966, 0.89915966]),
 'mean_test_score': array([0.8807423 , 0.87573529, 0.86992297, 0.87993697, 0.8682493 ,
        0.86656863, 0.86407563, 0.86740896, 0.86156863, 0.86990896,
        0.86656863, 0.86906863, 0.86739496, 0.86739496, 0.86822829,
        0.87156863, 0.87073529, 0.87156863, 0.87240896, 0.87157563,
        0.8732423 , 0.87490896, 0.87490896, 0.87408263, 0.87241597,
        0.87491597, 0.8732423 , 0.8732493 , 0.87240896, 0.8757493 ,
        0.87406863, 0.8732493 , 0.8757493 , 0.8757423 , 0.87741597,
        0.87491597, 0.87658263, 0.87408263, 0.87408263, 0.87741597,
        0.87741597, 0.87658263, 0.8782493 , 0.8782563 , 0.8757493 ,
        0.87908263, 0.87908263, 0.8782493 , 0.87658263, 0.87908263]),
 'std_test_score': array([0.02105329, 0.01949015, 0.02732994, 0.02321224, 0.02154497,
        0.01778378, 0.0209811 , 0.02447747, 0.01932915, 0.02038564,
        0.02493629, 0.02352783, 0.01948778, 0.01800606, 0.01775853,
        0.01864937, 0.0193935 , 0.0193798 , 0.0243824 , 0.02355348,
        0.02344172, 0.02166246, 0.02229439, 0.02046066, 0.02407217,
        0.02288317, 0.02159123, 0.02123999, 0.02101723, 0.01977728,
        0.02151447, 0.01881263, 0.02391017, 0.01591985, 0.01928669,
        0.01925777, 0.01882751, 0.01941576, 0.01868673, 0.01657571,
        0.01739345, 0.01646639, 0.01622001, 0.01782284, 0.01673409,
        0.01707926, 0.01707926, 0.01488026, 0.01420203, 0.01490827]),
 'rank_test_score': array([ 1, 19, 39,  2, 42, 47, 49, 44, 50, 40, 47, 41, 45, 45, 43, 36, 38,
        36, 33, 35, 30, 23, 22, 24, 32, 20, 30, 28, 33, 15, 27, 29, 16, 18,
         9, 20, 12, 24, 24,  9,  9, 13,  7,  6, 16,  3,  3,  7, 13,  3])}
```

```
In [40]: pred=clf_knn.predict(x_test)
         accuracy_score(pred,y_test)    # actual accuarcy on our test set.
```

```
Out[40]: 0.89
```

3. SUPPORT VECTOR MACHINE (SVM)

```
In [42]: params_dict={'C':[0.001,0.01,0.1,1,10,100],'gamma':[0.001,0.01,0.1,1,10,100],'kernel':['linear','rbf']}
         clf=GridSearchCV(estimator=SVC(),param_grid=params_dict,scoring='accuracy',cv=10)
         clf.fit(x_train,y_train)
```

```
Out[42]: GridSearchCV(cv=10, error_score='raise',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'C': [0.001, 0.01, 0.1, 1, 10, 100], 'gamma': [0.001, 0.01, 0.1, 1, 10, 100], 'kernel': ['li
near', 'rbf']},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='accuracy', verbose=0)
```

```
In [43]: clf.best_score_
```

```
Out[43]: 0.8999165971643036
```

```
In [44]: clf.best_params_
```

```
Out[44]: {'C': 10, 'gamma': 1, 'kernel': 'rbf'}
```

```
In [45]: # now tuning finally around these values of C and gamma and the kernel for further increasing the accuracy.
         params_dict={'C':[0.90,0.92,0.96,0.98,1.0,1.2,1.5],'gamma':[0.90,0.92,0.96,0.98,1.0,1.2,1.5],'kernel':['linear'
         clf_svm=GridSearchCV(estimator=SVC(),param_grid=params_dict,scoring='accuracy',cv=10)
         clf_svm.fit(x_train,y_train)
```

```
Out[45]: GridSearchCV(cv=10, error_score='raise',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'C': [0.9, 0.92, 0.96, 0.98, 1.0, 1.2, 1.5], 'gamma': [0.9, 0.92, 0.96, 0.98, 1.0, 1.2, 1.
5], 'kernel': ['linear', 'rbf']},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='accuracy', verbose=0)
```

```
In [46]: clf_svm.best_score_
```

```
Out[46]: 0.9065888240200167
```

```
In [47]: clf_svm.best_params_
```

```
Out[47]: {'C': 1.5, 'gamma': 0.9, 'kernel': 'rbf'}
```

```
In [48]: clf_svm.grid_scores_
```

```
C:\Users\HP\Anaconda3\lib\site-packages\sklearn\model_selection\_search.py:761: DeprecationWarning: The grid_sc
ores_ attribute was deprecated in version 0.18 in favor of the more elaborate cv_results_ attribute. The grid_s
cores_ attribute will not be available from 0.20
  DeprecationWarning)
```

```
Out[48]: [mean: 0.86322, std: 0.00307, params: {'C': 0.9, 'gamma': 0.9, 'kernel': 'linear'},
 mean: 0.89825, std: 0.01752, params: {'C': 0.9, 'gamma': 0.9, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 0.9, 'gamma': 0.92, 'kernel': 'linear'},
 mean: 0.89825, std: 0.01752, params: {'C': 0.9, 'gamma': 0.92, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 0.9, 'gamma': 0.96, 'kernel': 'linear'},
 mean: 0.89658, std: 0.01684, params: {'C': 0.9, 'gamma': 0.96, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 0.9, 'gamma': 0.98, 'kernel': 'linear'},
 mean: 0.89825, std: 0.01623, params: {'C': 0.9, 'gamma': 0.98, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 0.9, 'gamma': 1.0, 'kernel': 'linear'},
 mean: 0.89741, std: 0.01587, params: {'C': 0.9, 'gamma': 1.0, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 0.9, 'gamma': 1.2, 'kernel': 'linear'},
 mean: 0.89491, std: 0.01848, params: {'C': 0.9, 'gamma': 1.2, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 0.9, 'gamma': 1.5, 'kernel': 'linear'},
 mean: 0.89324, std: 0.01740, params: {'C': 0.9, 'gamma': 1.5, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 0.92, 'gamma': 0.9, 'kernel': 'linear'},
 mean: 0.89908, std: 0.01813, params: {'C': 0.92, 'gamma': 0.9, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 0.92, 'gamma': 0.92, 'kernel': 'linear'},
 mean: 0.89992, std: 0.01838, params: {'C': 0.92, 'gamma': 0.92, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 0.92, 'gamma': 0.96, 'kernel': 'linear'},
 mean: 0.89992, std: 0.01604, params: {'C': 0.92, 'gamma': 0.96, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 0.92, 'gamma': 0.98, 'kernel': 'linear'},
 mean: 0.89908, std: 0.01660, params: {'C': 0.92, 'gamma': 0.98, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 0.92, 'gamma': 1.0, 'kernel': 'linear'},
 mean: 0.89908, std: 0.01481, params: {'C': 0.92, 'gamma': 1.0, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 0.92, 'gamma': 1.2, 'kernel': 'linear'},
 mean: 0.89658, std: 0.01993, params: {'C': 0.92, 'gamma': 1.2, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 0.92, 'gamma': 1.5, 'kernel': 'linear'},
 mean: 0.89491, std: 0.01760, params: {'C': 0.92, 'gamma': 1.5, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 0.96, 'gamma': 0.9, 'kernel': 'linear'},
 mean: 0.89992, std: 0.01838, params: {'C': 0.96, 'gamma': 0.9, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 0.96, 'gamma': 0.92, 'kernel': 'linear'},
 mean: 0.90075, std: 0.01705, params: {'C': 0.96, 'gamma': 0.92, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 0.96, 'gamma': 0.96, 'kernel': 'linear'},
 mean: 0.89992, std: 0.01604, params: {'C': 0.96, 'gamma': 0.96, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 0.96, 'gamma': 0.98, 'kernel': 'linear'},
 mean: 0.89908, std: 0.01660, params: {'C': 0.96, 'gamma': 0.98, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 0.96, 'gamma': 1.0, 'kernel': 'linear'},
 mean: 0.89908, std: 0.01660, params: {'C': 0.96, 'gamma': 1.0, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 0.96, 'gamma': 1.2, 'kernel': 'linear'},
 mean: 0.89825, std: 0.01784, params: {'C': 0.96, 'gamma': 1.2, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 0.96, 'gamma': 1.5, 'kernel': 'linear'},
 mean: 0.89658, std: 0.01807, params: {'C': 0.96, 'gamma': 1.5, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 0.98, 'gamma': 0.9, 'kernel': 'linear'},
 mean: 0.89992, std: 0.01838, params: {'C': 0.98, 'gamma': 0.9, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 0.98, 'gamma': 0.92, 'kernel': 'linear'},
 mean: 0.89992, std: 0.01838, params: {'C': 0.98, 'gamma': 0.92, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 0.98, 'gamma': 0.96, 'kernel': 'linear'},
 mean: 0.89992, std: 0.01604, params: {'C': 0.98, 'gamma': 0.96, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 0.98, 'gamma': 0.98, 'kernel': 'linear'},
 mean: 0.89908, std: 0.01660, params: {'C': 0.98, 'gamma': 0.98, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 0.98, 'gamma': 1.0, 'kernel': 'linear'},
 mean: 0.89908, std: 0.01660, params: {'C': 0.98, 'gamma': 1.0, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 0.98, 'gamma': 1.2, 'kernel': 'linear'},
 mean: 0.89825, std: 0.01784, params: {'C': 0.98, 'gamma': 1.2, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 0.98, 'gamma': 1.5, 'kernel': 'linear'},
 mean: 0.89658, std: 0.01807, params: {'C': 0.98, 'gamma': 1.5, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 1.0, 'gamma': 0.9, 'kernel': 'linear'},
 mean: 0.90075, std: 0.01926, params: {'C': 1.0, 'gamma': 0.9, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 1.0, 'gamma': 0.92, 'kernel': 'linear'},
 mean: 0.89992, std: 0.01838, params: {'C': 1.0, 'gamma': 0.92, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 1.0, 'gamma': 0.96, 'kernel': 'linear'},
 mean: 0.89992, std: 0.01604, params: {'C': 1.0, 'gamma': 0.96, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 1.0, 'gamma': 0.98, 'kernel': 'linear'},
 mean: 0.89992, std: 0.01604, params: {'C': 1.0, 'gamma': 0.98, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 1.0, 'gamma': 1.0, 'kernel': 'linear'},
 mean: 0.89908, std: 0.01660, params: {'C': 1.0, 'gamma': 1.0, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 1.0, 'gamma': 1.2, 'kernel': 'linear'},
 mean: 0.89992, std: 0.01716, params: {'C': 1.0, 'gamma': 1.2, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 1.0, 'gamma': 1.5, 'kernel': 'linear'},
 mean: 0.89658, std: 0.01807, params: {'C': 1.0, 'gamma': 1.5, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 1.2, 'gamma': 0.9, 'kernel': 'linear'},
 mean: 0.90492, std: 0.02427, params: {'C': 1.2, 'gamma': 0.9, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 1.2, 'gamma': 0.92, 'kernel': 'linear'},
 mean: 0.90492, std: 0.02427, params: {'C': 1.2, 'gamma': 0.92, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 1.2, 'gamma': 0.96, 'kernel': 'linear'},
 mean: 0.90492, std: 0.02252, params: {'C': 1.2, 'gamma': 0.96, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 1.2, 'gamma': 0.98, 'kernel': 'linear'},
 mean: 0.90492, std: 0.02252, params: {'C': 1.2, 'gamma': 0.98, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 1.2, 'gamma': 1.0, 'kernel': 'linear'},
 mean: 0.90409, std: 0.02128, params: {'C': 1.2, 'gamma': 1.0, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 1.2, 'gamma': 1.2, 'kernel': 'linear'},
 mean: 0.90325, std: 0.01690, params: {'C': 1.2, 'gamma': 1.2, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 1.2, 'gamma': 1.5, 'kernel': 'linear'},
 mean: 0.90075, std: 0.01532, params: {'C': 1.2, 'gamma': 1.5, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 1.5, 'gamma': 0.9, 'kernel': 'linear'},
 mean: 0.90659, std: 0.02143, params: {'C': 1.5, 'gamma': 0.9, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 1.5, 'gamma': 0.92, 'kernel': 'linear'},
 mean: 0.90659, std: 0.02143, params: {'C': 1.5, 'gamma': 0.92, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 1.5, 'gamma': 0.96, 'kernel': 'linear'},
 mean: 0.90659, std: 0.02121, params: {'C': 1.5, 'gamma': 0.96, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 1.5, 'gamma': 0.98, 'kernel': 'linear'},
 mean: 0.90575, std: 0.02251, params: {'C': 1.5, 'gamma': 0.98, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 1.5, 'gamma': 1.0, 'kernel': 'linear'},
 mean: 0.90575, std: 0.02251, params: {'C': 1.5, 'gamma': 1.0, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 1.5, 'gamma': 1.2, 'kernel': 'linear'},
 mean: 0.90409, std: 0.01961, params: {'C': 1.5, 'gamma': 1.2, 'kernel': 'rbf'},
 mean: 0.86322, std: 0.00307, params: {'C': 1.5, 'gamma': 1.5, 'kernel': 'linear'},
 mean: 0.90325, std: 0.01647, params: {'C': 1.5, 'gamma': 1.5, 'kernel': 'rbf'}]
```

```
In [49]: pred=clf_svm.predict(x_test)
         accuracy_score(pred,y_test)    # actual accuarcy on our test set.
```

```
Out[49]: 0.9175
```

HENCE TILL NOW THE BEST ACCURACY IS GIVEN BY SVM WITH rbf KERNEL WITH C=1.5 and gamma=0.90 .

4. RANDOM FOREST.

```
In [41]: params_dict={'n_estimators':[500],'max_features':['auto','sqrt','log2']}
         clf_rf=GridSearchCV(estimator=RandomForestClassifier(n_jobs=-1),param_grid=params_dict,scoring='accuracy',cv=10
         clf_rf.fit(x_train,y_train)
```

```
Out[41]: ▸ GridSearchCV
         ▸ estimator: RandomForestClassifier
             ▸ RandomForestClassifier
```

```
In [42]: clf_rf.best_score_
```

```
Out[42]: 0.9107843137254902
```

```
In [43]: clf_rf.best_params_
```

```
Out[43]: {'max_features': 'log2', 'n_estimators': 500}
```

```
In [44]: clf_rf.cv_results_
```

```
Out[44]: {'mean_fit_time': array([3.99804115e-04, 1.12589784e+00, 1.16392264e+00]),
 'std_fit_time': array([0.00048966, 0.01183291, 0.06009783]),
 'mean_score_time': array([0.        , 0.08079278, 0.09153562]),
 'std_score_time': array([0.        , 0.00048326, 0.02844147]),
 'param_max_features': masked_array(data=['auto', 'sqrt', 'log2'],
              mask=[False, False, False],
        fill_value='?',
             dtype=object),
 'param_n_estimators': masked_array(data=[500, 500, 500],
              mask=[False, False, False],
        fill_value='?',
             dtype=object),
 'params': [{'max_features': 'auto', 'n_estimators': 500},
  {'max_features': 'sqrt', 'n_estimators': 500},
  {'max_features': 'log2', 'n_estimators': 500}],
 'split0_test_score': array([       nan, 0.88333333, 0.875     ]),
 'split1_test_score': array([       nan, 0.9       , 0.90833333]),
 'split2_test_score': array([  nan, 0.9  , 0.925]),
 'split3_test_score': array([       nan, 0.93333333, 0.93333333]),
 'split4_test_score': array([       nan, 0.89166667, 0.89166667]),
 'split5_test_score': array([  nan, 0.875, 0.875]),
 'split6_test_score': array([       nan, 0.93333333, 0.925     ]),
 'split7_test_score': array([       nan, 0.94166667, 0.93333333]),
 'split8_test_score': array([nan, 0.9, 0.9]),
 'split9_test_score': array([       nan, 0.93277311, 0.94117647]),
 'mean_test_score': array([       nan, 0.90911064, 0.91078431]),
 'std_test_score': array([       nan, 0.02274773, 0.02322412]),
 'rank_test_score': array([3, 2, 1])}
```

```
In [45]: pred=clf_rf.predict(x_test)
         accuracy_score(pred,y_test)    # actual accuarcy on our test set.
```

```
Out[45]: 0.91
```

5. GRADIENT BOOSTING.

```
In [46]: clf_gb=GridSearchCV(estimator=GradientBoostingClassifier(),cv=10,param_grid=dict({'n_estimators':[500]}))
         clf_gb.fit(x_train,y_train)
```

```
Out[46]: ▸ GridSearchCV
         ▸ estimator: GradientBoostingClassifier
             ▸ GradientBoostingClassifier
```

```
In [47]: clf_gb.best_score_
```

```
Out[47]: 0.9082703081232493
```

```
In [48]: clf_gb.best_params_
```

```
Out[48]: {'n_estimators': 500}
```

```
In [49]: pred=clf_gb.predict(x_test)
         accuracy_score(pred,y_test)
```

```
Out[49]: 0.89
```

HENCE ON GIVEN PARAMETER TUNING THE SVM WITH rbf KERNEL GIVES THE HIGHEST ACCURACY OF 91.75%

```
In [ ]:
```