In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

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

Out[2]:

| | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | pH | sulphates | alcohol |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 7.4 | 0.700 | 0.00 | 1.9 | 0.076 | 11.0 | 34.0 | 0.99780 | 3.51 | 0.56 | 9.4 |
| **1** | 7.8 | 0.880 | 0.00 | 2.6 | 0.098 | 25.0 | 67.0 | 0.99680 | 3.20 | 0.68 | 9.8 |
| **2** | 7.8 | 0.760 | 0.04 | 2.3 | 0.092 | 15.0 | 54.0 | 0.99700 | 3.26 | 0.65 | 9.8 |
| **3** | 11.2 | 0.280 | 0.56 | 1.9 | 0.075 | 17.0 | 60.0 | 0.99800 | 3.16 | 0.58 | 9.8 |
| **4** | 7.4 | 0.700 | 0.00 | 1.9 | 0.076 | 11.0 | 34.0 | 0.99780 | 3.51 | 0.56 | 9.4 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1594** | 6.2 | 0.600 | 0.08 | 2.0 | 0.090 | 32.0 | 44.0 | 0.99490 | 3.45 | 0.58 | 10.5 |
| **1595** | 5.9 | 0.550 | 0.10 | 2.2 | 0.062 | 39.0 | 51.0 | 0.99512 | 3.52 | 0.76 | 11.2 |
| **1596** | 6.3 | 0.510 | 0.13 | 2.3 | 0.076 | 29.0 | 40.0 | 0.99574 | 3.42 | 0.75 | 11.0 |
| **1597** | 5.9 | 0.645 | 0.12 | 2.0 | 0.075 | 32.0 | 44.0 | 0.99547 | 3.57 | 0.71 | 10.2 |
| **1598** | 6.0 | 0.310 | 0.47 | 3.6 | 0.067 | 18.0 | 42.0 | 0.99549 | 3.39 | 0.66 | 11.0 |

1599 rows × 12 columns

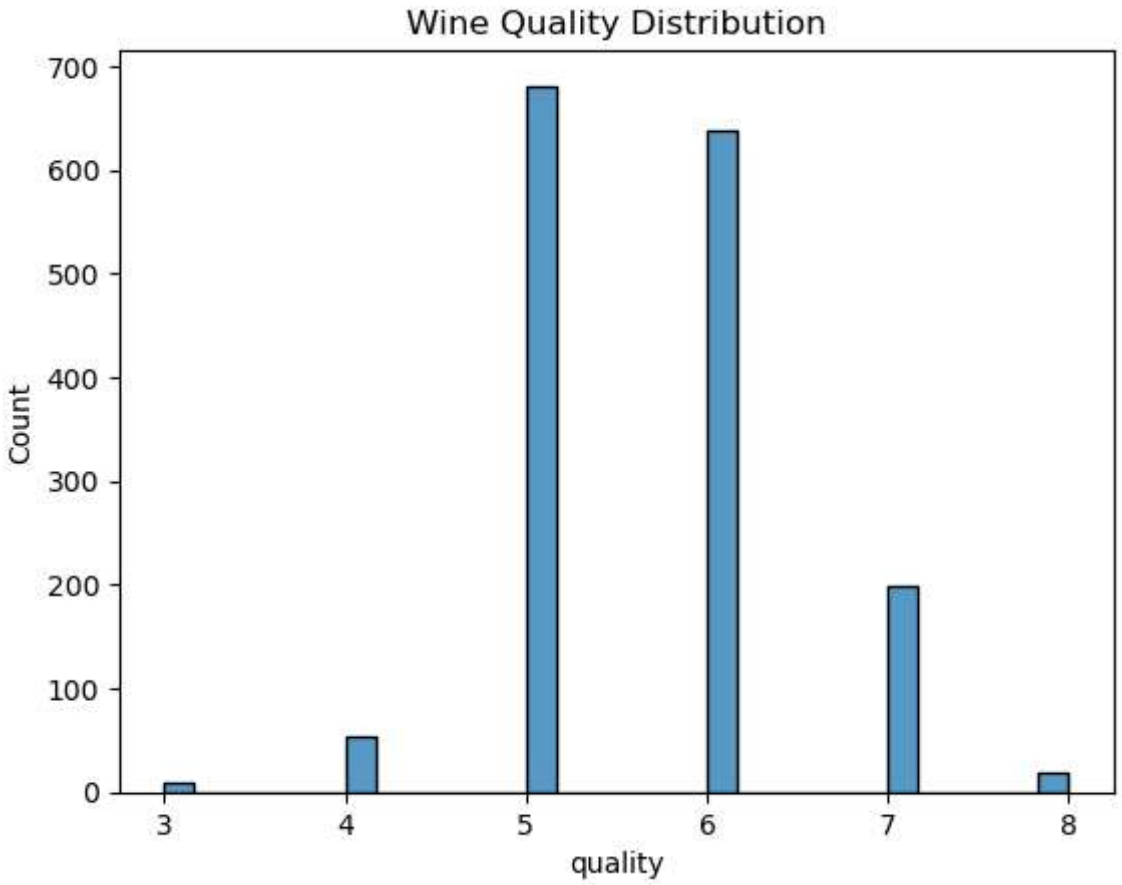
In [3]:
```
df.isnull().sum()
```

Out[3]:
```
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 [4]:
```
display(list(df))
sns.histplot(x=df.quality)
plt.title("Wine Quality Distribution")
```

```
['fixed acidity',
 'volatile acidity',
 'citric acid',
 'residual sugar',
 'chlorides',
 'free sulfur dioxide',
 'total sulfur dioxide',
 'density',
 'pH',
 'sulphates',
 'alcohol',
 'quality']
```

Out[4]: Text(0.5, 1.0, 'Wine Quality Distribution')




In [5]:
```
df.describe()
```

Out[5]:

| | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | 15 |
|---|---|---|---|---|---|---|---|---|
| **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 [6]:
```
df.shape
```

Out[6]:
```
(1599, 12)
```

In [7]:
```
df.duplicated().sum()
```

Out[7]:
```
240
```

In [8]:
```
df.drop_duplicates(keep = 'first',inplace = True, ignore_index = True)
df.shape
```

Out[8]:
```
(1359, 12)
```

In [9]:
```
df['quality'].value_counts()
```

Out[9]:
```
quality
5    577
6    535
7    167
4     53
8     17
3     10
Name: count, dtype: int64
```

In [10]:
```
df['Class'] = np.where(df['quality']>=7,1,0)
```

In [11]:
```
df
```

Out[11]:

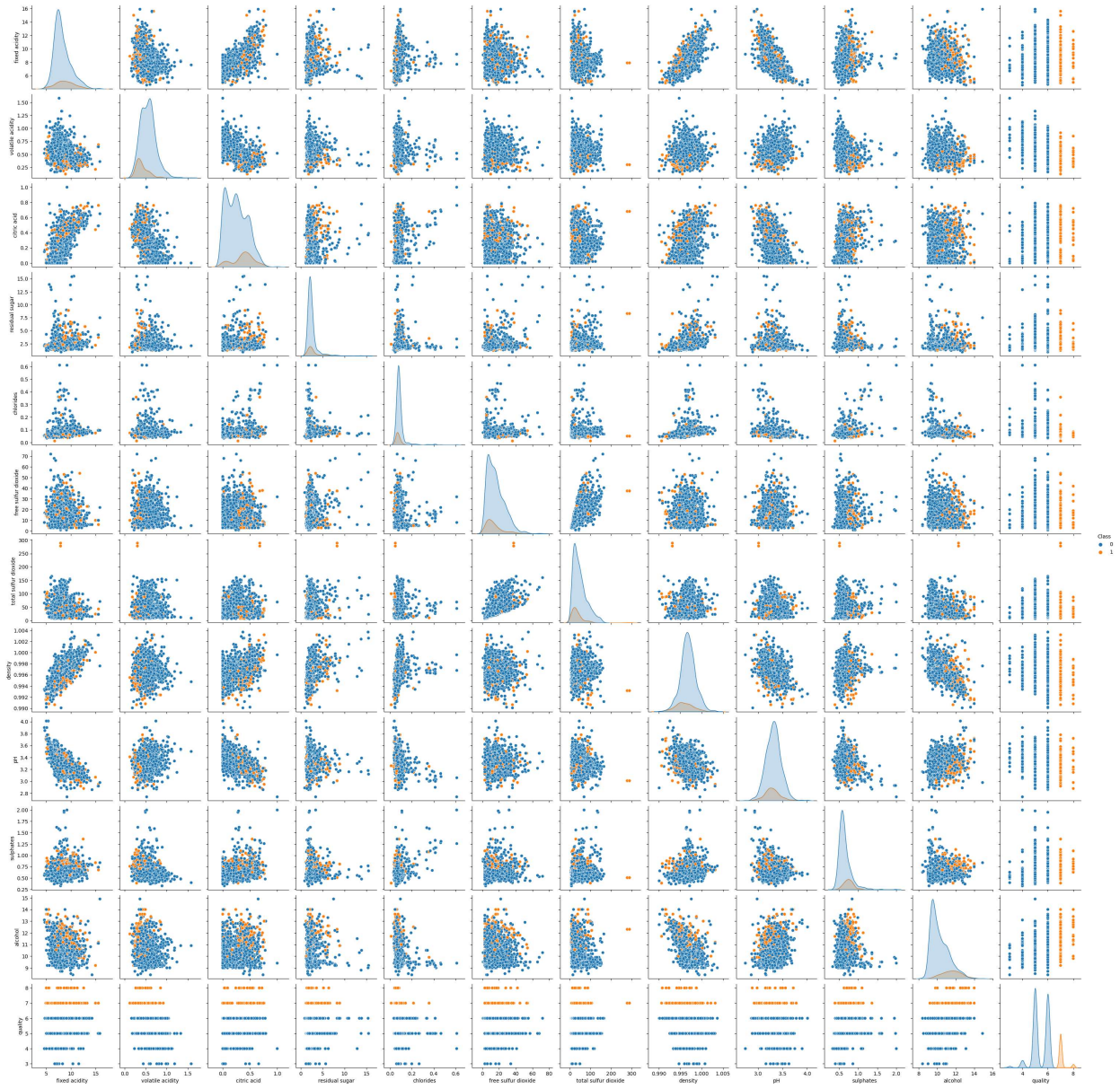
| | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | pH | sulphates | alcohol |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 7.4 | 0.700 | 0.00 | 1.9 | 0.076 | 11.0 | 34.0 | 0.99780 | 3.51 | 0.56 | 9.4 |
| **1** | 7.8 | 0.880 | 0.00 | 2.6 | 0.098 | 25.0 | 67.0 | 0.99680 | 3.20 | 0.68 | 9.8 |
| **2** | 7.8 | 0.760 | 0.04 | 2.3 | 0.092 | 15.0 | 54.0 | 0.99700 | 3.26 | 0.65 | 9.8 |
| **3** | 11.2 | 0.280 | 0.56 | 1.9 | 0.075 | 17.0 | 60.0 | 0.99800 | 3.16 | 0.58 | 9.8 |
| **4** | 7.4 | 0.660 | 0.00 | 1.8 | 0.075 | 13.0 | 40.0 | 0.99780 | 3.51 | 0.56 | 9.4 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1354** | 6.8 | 0.620 | 0.08 | 1.9 | 0.068 | 28.0 | 38.0 | 0.99651 | 3.42 | 0.82 | 9.5 |
| **1355** | 6.2 | 0.600 | 0.08 | 2.0 | 0.090 | 32.0 | 44.0 | 0.99490 | 3.45 | 0.58 | 10.5 |
| **1356** | 5.9 | 0.550 | 0.10 | 2.2 | 0.062 | 39.0 | 51.0 | 0.99512 | 3.52 | 0.76 | 11.2 |
| **1357** | 5.9 | 0.645 | 0.12 | 2.0 | 0.075 | 32.0 | 44.0 | 0.99547 | 3.57 | 0.71 | 10.2 |
| **1358** | 6.0 | 0.310 | 0.47 | 3.6 | 0.067 | 18.0 | 42.0 | 0.99549 | 3.39 | 0.66 | 11.0 |

1359 rows × 13 columns

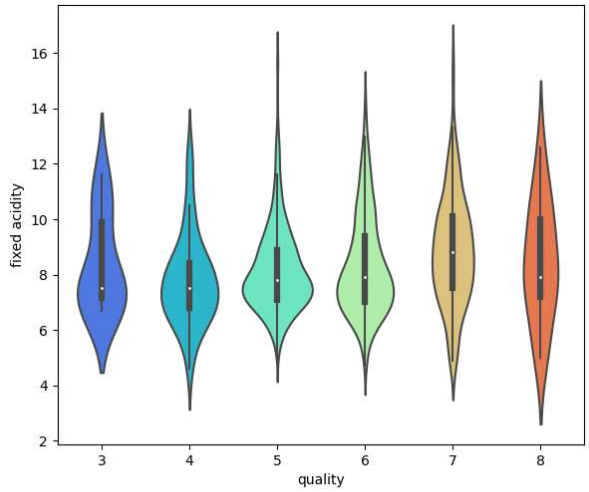
In [12]:
```
plt.figure(figsize = (15,12))
sns.heatmap(df.corr(), annot = True, cmap = 'Pastel1')
plt.show()
```

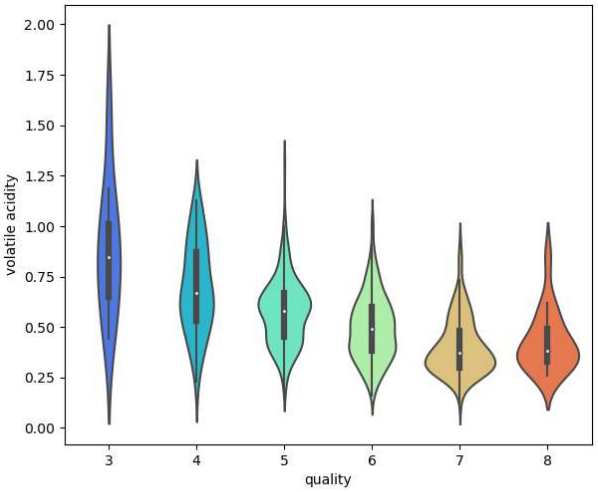

In [13]:
```
plt.figure(figsize = (15,20))
sns.pairplot(data=df, hue='Class')
plt.show()
```

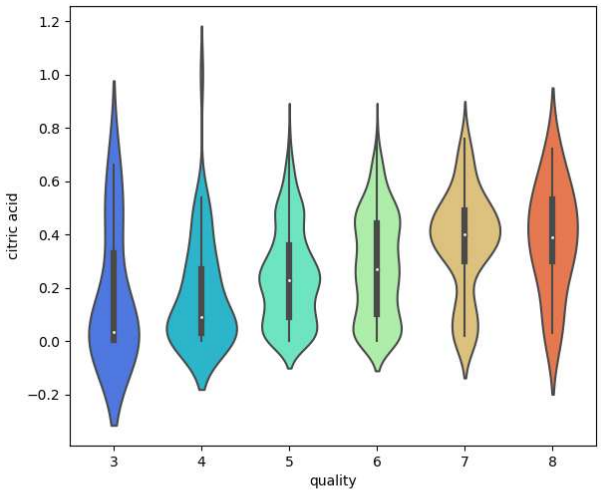
```
<Figure size 1500x2000 with 0 Axes>
```

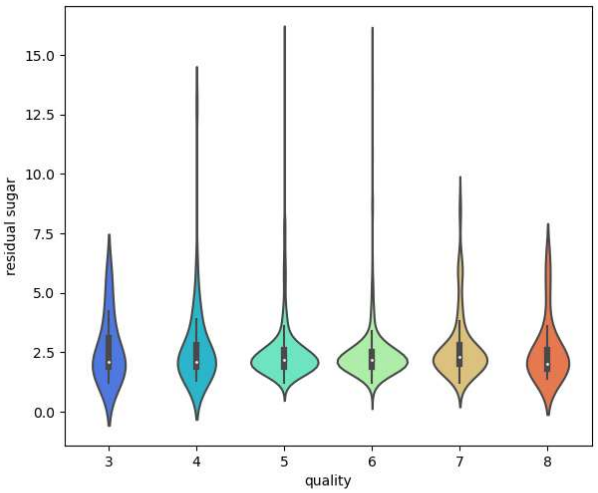


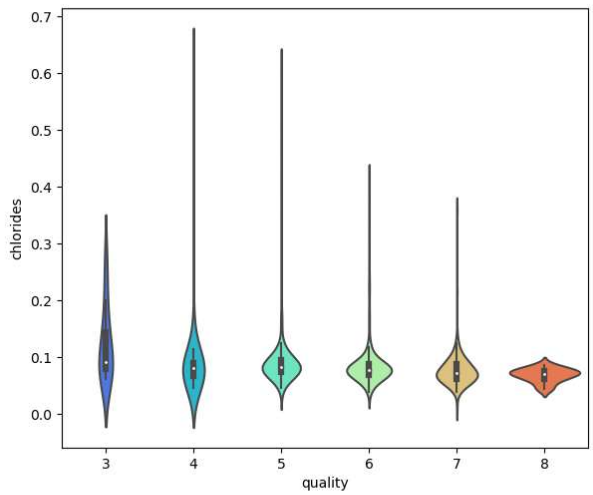

In [14]:
```
plt.figure(figsize=(15,150))
plot_number = 1
for i in list(df.select_dtypes(include=['float']).columns)[0:]:
    plt.subplot(22,2,plot_number)
    sns.violinplot(x="quality", y = i, data=df,palette='rainbow')
    plot_number+=1
plt.show()
```

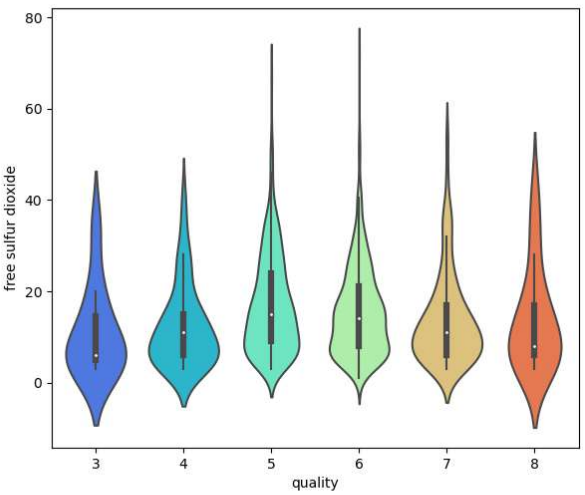

fixed acidity
quality

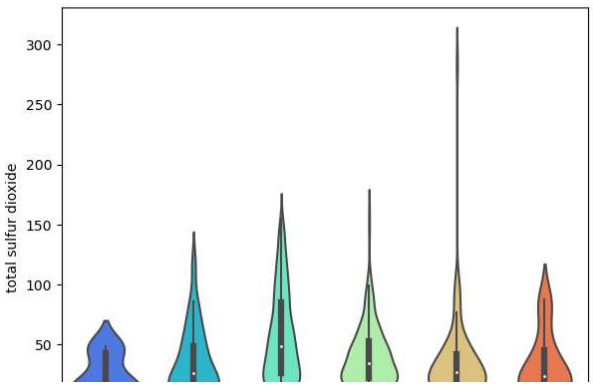

volatile acidity
quality

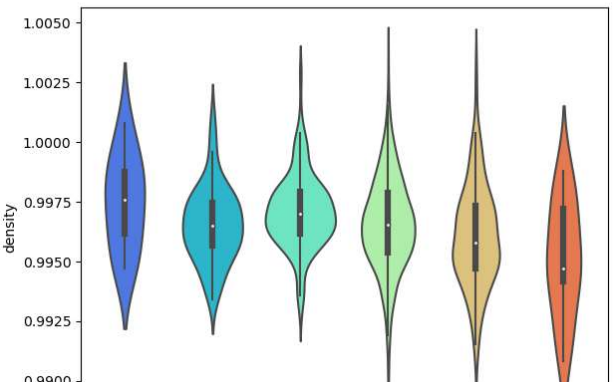

citric acid
quality

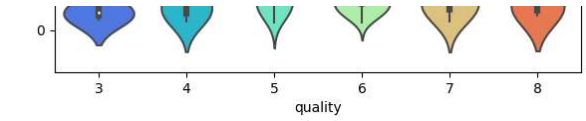

residual sugar
quality

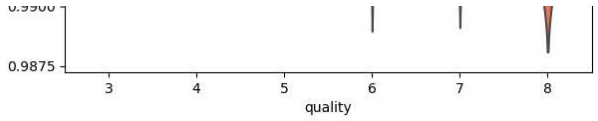

chlorides
quality

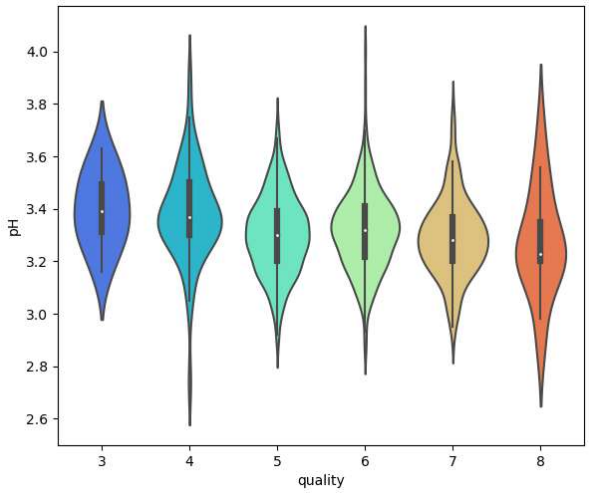

free sulfur dioxide
quality

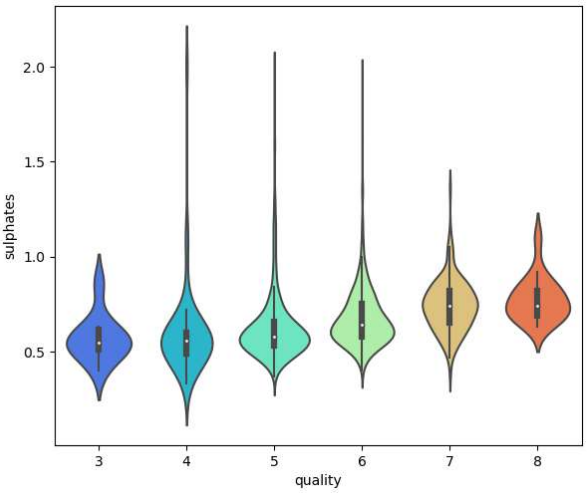

total sulfur dioxide

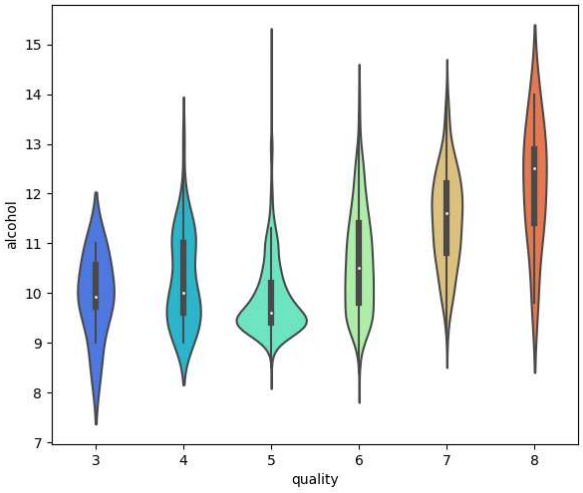

density



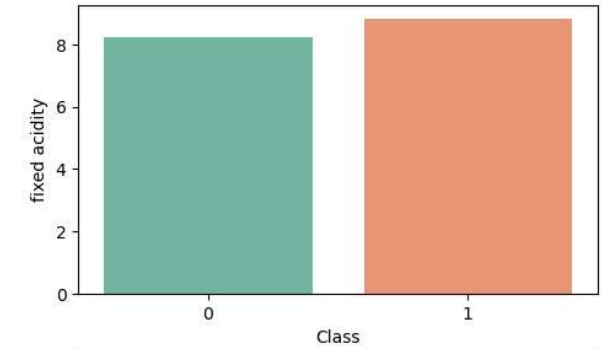



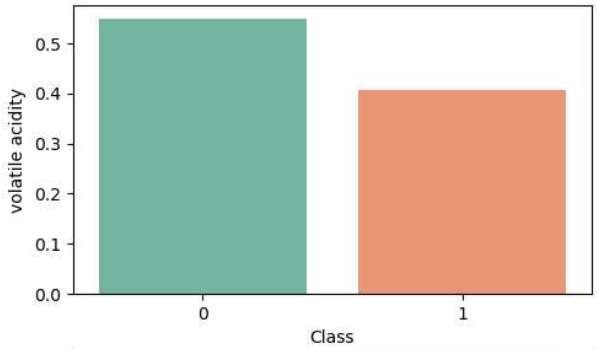

In [15]:
```
cmp = df.groupby('Class').mean()
cmp
```

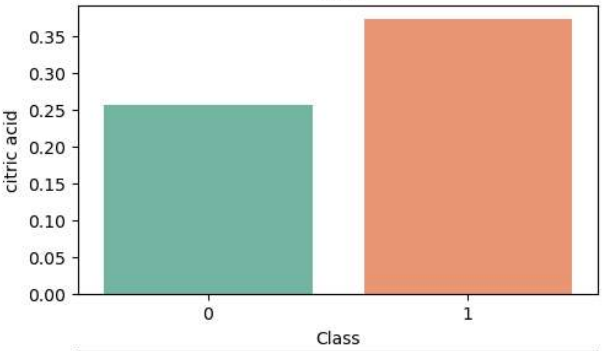
Out[15]:

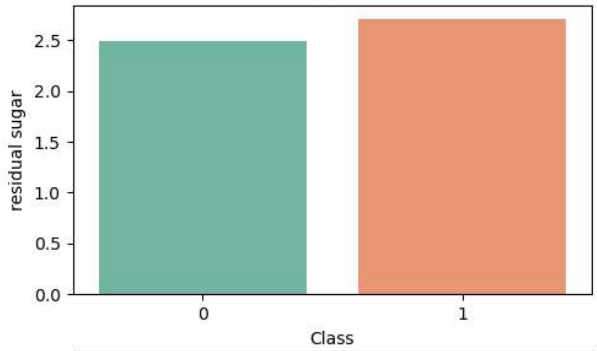
| Class | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | pH | sul |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 8.230809 | 0.548834 | 0.256587 | 2.495149 | 0.090014 | 16.194043 | 48.621702 | 0.996826 | 3.313106 | 0.6 |
| 1 | 8.820109 | 0.405870 | 0.372880 | 2.703804 | 0.076049 | 13.972826 | 35.358696 | 0.995963 | 3.288587 | 0.7 |

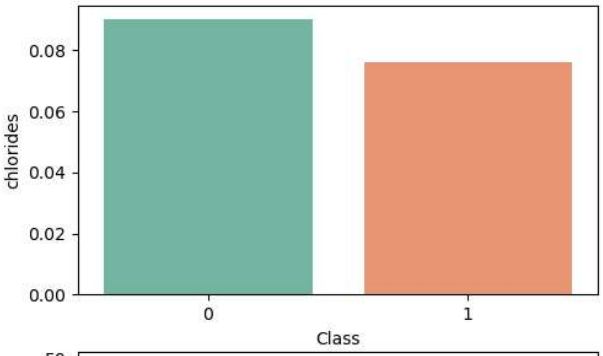
In [16]:
```
plt.figure(figsize = (12,80))
plt_num = 1
for i in cmp.columns:
    plt.subplot(22,2,plt_num)
    sns.barplot(x = cmp.index, y = cmp[i], palette ='Set2')
    plt_num+=1
plt.show()
```

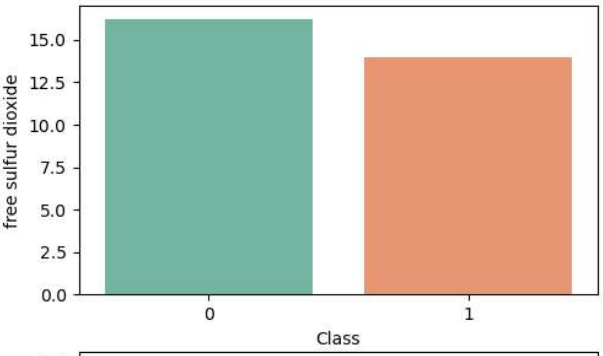

fixed acidity
Class

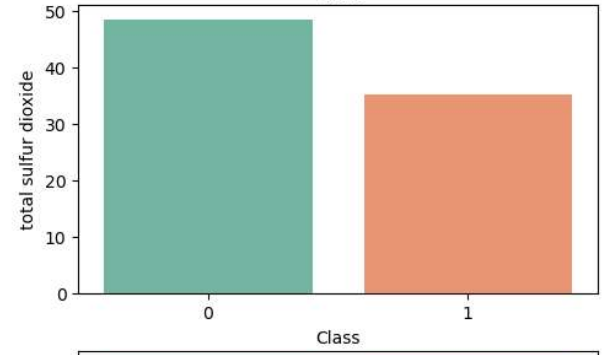

volatile acidity
Class

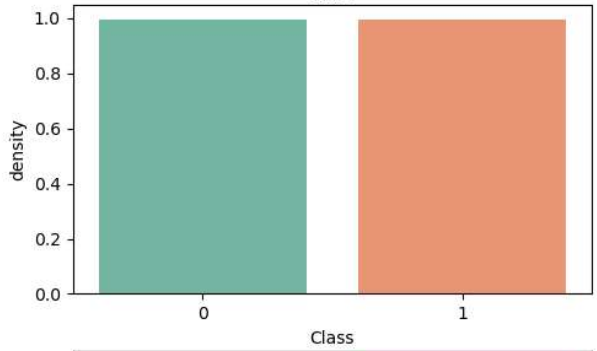

citric acid
Class

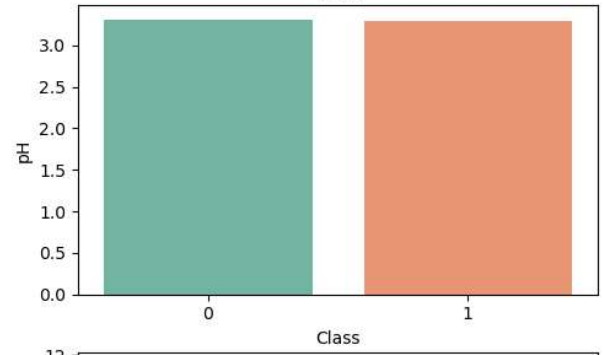

residual sugar
Class

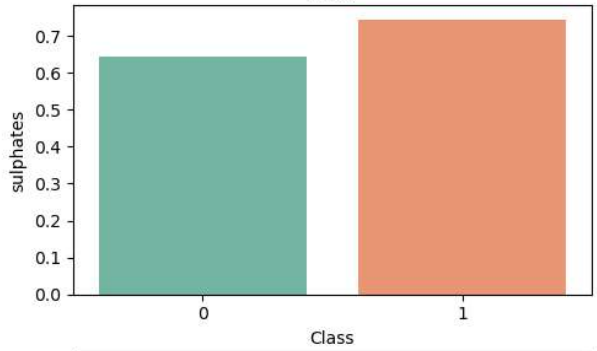

chlorides
Class


free sulfur dioxide
Class


total sulfur dioxide
Class


density
Class


pH
Class


sulphates
Class


alcohol


quality



In [17]:
```
df['Class'].value_counts()
```

Out[17]:
```
Class
0    1175
1     184
Name: count, dtype: int64
```

In [18]:
```
good = df[df['Class']==1]
bad = df[df['Class']==0]

good = good.sample(bad.shape[0], replace=True)

print(bad.shape)
print(good.shape)
```

```
(1175, 13)
(1175, 13)
```

In [19]:
```
data = pd.concat([bad, good])
data.shape
```

Out[19]:
```
(2350, 13)
```

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

In [21]:
```
data.head()
```

Out[21]:

| | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | pH | sulphates | alcohol | qua |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 | 0.66 | 0.00 | 1.8 | 0.075 | 13.0 | 40.0 | 0.9978 | 3.51 | 0.56 | 9.4 | |

In [22]:
```
x = data.drop('Class', axis = 1)
y = data['Class']
```

In [23]:
```
x.head()
```

Out[23]:

| | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | pH | sulphates | alcohol | qua |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **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 | 0.66 | 0.00 | 1.8 | 0.075 | 13.0 | 40.0 | 0.9978 | 3.51 | 0.56 | 9.4 | |

In [24]:
```
y.head()
```

Out[24]:
```
0    0
1    0
2    0
3    0
4    0
Name: Class, dtype: int32
```

In [25]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_scaler = scaler.fit_transform(x)
```

In [26]:
```
pd.DataFrame(x_scaler, columns = x.columns).head()
```

Out[26]:

| | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | pH | sul |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | -0.622021 | 1.241914 | -1.562028 | -0.516303 | -0.167332 | -0.398335 | -0.235469 | 0.651899 | 1.338337 | -0.8 |
| **1** | -0.410403 | 2.253807 | -1.562028 | -0.021953 | 0.344383 | 0.940503 | 0.736227 | 0.160717 | -0.632870 | -0. |
| **2** | -0.410403 | 1.579211 | -1.364058 | -0.233817 | 0.204824 | -0.015810 | 0.353438 | 0.258953 | -0.251346 | -0. |
| **3** | 1.388347 | -1.119171 | 1.209557 | -0.516303 | -0.190591 | 0.175453 | 0.530110 | 0.750136 | -0.887219 | -0. |
| **4** | -0.622021 | 1.017048 | -1.562028 | -0.586925 | -0.190591 | -0.207072 | -0.058797 | 0.651899 | 1.338337 | -0.8 |

In [27]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train,y_test = train_test_split(x_scaler, y, test_size = 0.25, stra
```

In [28]:
```
print(x_train.shape, x_test.shape, y_train.shape,y_test.shape)
```

```
(1762, 12) (588, 12) (1762,) (588,)
```

In [29]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(x_train,y_train)
```

Out[29]:
```
LogisticRegression
LogisticRegression()
```

In [30]:
```
train_pred_lr = lr.predict(x_train)
test_pred_lr = lr.predict(x_test)
```

In [31]:
```
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
print(confusion_matrix(y_train,train_pred_lr))
print()
print(confusion_matrix(y_test,test_pred_lr))
```

```
[[881   0]
 [  0 881]]

[[294   0]
 [  0 294]]
```

In [32]:
```
print(classification_report(y_train,train_pred_lr))
print()
print(classification_report(y_test,test_pred_lr))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       881
           1       1.00      1.00      1.00       881

    accuracy                           1.00      1762
   macro avg       1.00      1.00      1.00      1762
weighted avg       1.00      1.00      1.00      1762


              precision    recall  f1-score   support

           0       1.00      1.00      1.00       294
           1       1.00      1.00      1.00       294

    accuracy                           1.00       588
   macro avg       1.00      1.00      1.00       588
weighted avg       1.00      1.00      1.00       588
```

In [33]:
```
train_accu = []
test_accu = []
train_mean = []
test_mean = []
```

In [34]:
```
print(accuracy_score(y_train,train_pred_lr))
train_accu.append(accuracy_score(y_train,train_pred_lr))
print()
print(accuracy_score(y_test,test_pred_lr))
test_accu.append(accuracy_score(y_test,test_pred_lr))
```

```
1.0

1.0
```

In [35]:
```
from sklearn.model_selection import cross_val_score
training_accuracy = cross_val_score(lr, x_train, y_train, cv=10)
```

```
test_accuracy = cross_val_score(lr, x_test, y_test, cv=10)
print("Train Accuracy", training_accuracy)
print()
print("Train Mean Accuracy", training_accuracy.mean())
print()
print("Train Max Accuracy", training_accuracy.max())
print()
print("Test Accuracy", test_accuracy)
print()
print("Test Mean Accuracy", test_accuracy.mean())
print()
print("Test Max Accuracy", test_accuracy.max())
train_mean.append(training_accuracy.mean())
test_mean.append(test_accuracy.mean())
```

```
Train Accuracy [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

Train Mean Accuracy 1.0

Train Max Accuracy 1.0

Test Accuracy [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

Test Mean Accuracy 1.0

Test Max Accuracy 1.0
```

In [36]:
```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(max_depth = 8,min_samples_leaf = 2)
dtree.fit(x_train,y_train)
```

Out[36]:
```
DecisionTreeClassifier(max_depth=8, min_samples_leaf=2)
```

In [37]:
```
train_pred_dtree = dtree.predict(x_train)
test_pred_dtree = dtree.predict(x_test)
```

In [38]:
```
print(confusion_matrix(y_train,train_pred_dtree))
print()
print(confusion_matrix(y_test,test_pred_dtree))
```

```
[[881   0]
 [  0 881]]

[[294   0]
 [  0 294]]
```

In [39]:
```
print(classification_report(y_train,train_pred_dtree))
print()
print(classification_report(y_test,test_pred_dtree))
```

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 881 |
| 1 | 1.00 | 1.00 | 1.00 | 881 |
| accuracy | | | 1.00 | 1762 |
| macro avg | 1.00 | 1.00 | 1.00 | 1762 |
| weighted avg | 1.00 | 1.00 | 1.00 | 1762 |

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 294 |
| 1 | 1.00 | 1.00 | 1.00 | 294 |
| accuracy | | | 1.00 | 588 |
| macro avg | 1.00 | 1.00 | 1.00 | 588 |
| weighted avg | 1.00 | 1.00 | 1.00 | 588 |

In [40]:
```
print(accuracy_score(y_train,train_pred_dtree))
train_accu.append(accuracy_score(y_train,train_pred_dtree))
print()
print(accuracy_score(y_test,test_pred_dtree))
test_accu.append(accuracy_score(y_test,test_pred_dtree))
```

```
1.0

1.0
```

In [41]:
```
from sklearn.model_selection import cross_val_score
training_accuracy = cross_val_score(dtree,x_train, y_train, cv=10)
test_accuracy = cross_val_score(dtree,x_train, y_train, cv=10)
print("Train Accuracy", training_accuracy)
print()
print("Train Mean Accuracy", training_accuracy.mean())
print()
print("Train Max Accuracy", training_accuracy.max())
print()
print("Test Accuracy", test_accuracy)
print()
print("Test Mean Accuracy", test_accuracy.mean())
print()
print("Test Max Accuracy", test_accuracy.max())
train_mean.append(training_accuracy.mean())
test_mean.append(test_accuracy.mean())
```

```
Train Accuracy [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

Train Mean Accuracy 1.0

Train Max Accuracy 1.0

Test Accuracy [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

Test Mean Accuracy 1.0

Test Max Accuracy 1.0
```

In [42]:
```
from sklearn.ensemble import RandomForestClassifier
rfr = RandomForestClassifier(n_estimators=200, max_depth=8, min_samples_leaf= 1, min_s
```

```
rfr.fit(x_train,y_train)
```

Out[42]:

RandomForestClassifier

```
RandomForestClassifier(max_depth=8, n_estimators=200, random_state=200)
```

In [43]:

```
train_pred_rfr = rfr.predict(x_train)
test_pred_rfr = rfr.predict(x_test)

print(confusion_matrix(y_train,train_pred_rfr))
print()
print(confusion_matrix(y_test,test_pred_rfr))
```

```
[[881   0]
 [  0 881]]

[[294   0]
 [  0 294]]
```

In [44]:

```
print(classification_report(y_train,train_pred_rfr))
print()
print(classification_report(y_test,test_pred_rfr))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       881
           1       1.00      1.00      1.00       881

    accuracy                           1.00      1762
   macro avg       1.00      1.00      1.00      1762
weighted avg       1.00      1.00      1.00      1762


              precision    recall  f1-score   support

           0       1.00      1.00      1.00       294
           1       1.00      1.00      1.00       294

    accuracy                           1.00       588
   macro avg       1.00      1.00      1.00       588
weighted avg       1.00      1.00      1.00       588
```

In [45]:

```
print(accuracy_score(y_train,train_pred_rfr))
train_accu.append(accuracy_score(y_train,train_pred_rfr))
print()
print(accuracy_score(y_test,test_pred_rfr))
test_accu.append(accuracy_score(y_test,test_pred_rfr))
```

```
1.0

1.0
```

In [46]:

```
from sklearn.model_selection import cross_val_score
training_accuracy = cross_val_score(rfr, x_train, y_train, cv=10)
test_accuracy = cross_val_score(rfr, x_test, y_test, cv=10)
print("Train Accuracy", training_accuracy)
print()
print("Train Mean Accuracy", training_accuracy.mean())
print()
```

```
print("Train Max Accuracy", training_accuracy.max())
print()
print("Test Accuracy", test_accuracy)
print()
print("Test Mean Accuracy", test_accuracy.mean())
print()
print("Test Max Accuracy", test_accuracy.max())
train_mean.append(training_accuracy.mean())
test_mean.append(test_accuracy.mean())
```

```
Train Accuracy [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

Train Mean Accuracy 1.0

Train Max Accuracy 1.0

Test Accuracy [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

Test Mean Accuracy 1.0

Test Max Accuracy 1.0
```

In [47]:
```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\prane\anaconda3\lib\site-packages
(2.0.2)
Requirement already satisfied: numpy in c:\users\prane\anaconda3\lib\site-packages (f
rom xgboost) (1.24.3)
Requirement already satisfied: scipy in c:\users\prane\anaconda3\lib\site-packages (f
rom xgboost) (1.11.1)
WARNING: There was an error checking the latest version of pip.
```

In [48]:
```
from xgboost import XGBClassifier
xgb = XGBClassifier()
xgb.fit(x_train,y_train)
```

Out[48]:

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=N
one,
              enable_categorical=False, eval_metric=None, feature_types=N
one,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=N
one,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
```

In [49]:
```
train_pred_xgb = xgb.predict(x_train)
test_pred_xgb = xgb.predict(x_test)

print(confusion_matrix(y_train,train_pred_xgb))
print()
print(confusion_matrix(y_test,test_pred_xgb))
```

```
[[881   0]
 [  0 881]]

[[294   0]
 [  0 294]]
```

In [50]:
```
print(classification_report(y_train,train_pred_xgb))
print()
print(classification_report(y_test,test_pred_xgb))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       881
           1       1.00      1.00      1.00       881

    accuracy                           1.00      1762
   macro avg       1.00      1.00      1.00      1762
weighted avg       1.00      1.00      1.00      1762


              precision    recall  f1-score   support

           0       1.00      1.00      1.00       294
           1       1.00      1.00      1.00       294

    accuracy                           1.00       588
   macro avg       1.00      1.00      1.00       588
weighted avg       1.00      1.00      1.00       588
```

In [51]:
```
print(accuracy_score(y_train,train_pred_xgb))
train_accu.append(accuracy_score(y_train,train_pred_xgb))
print()
print(accuracy_score(y_test,test_pred_xgb))
test_accu.append(accuracy_score(y_test,test_pred_xgb))
```

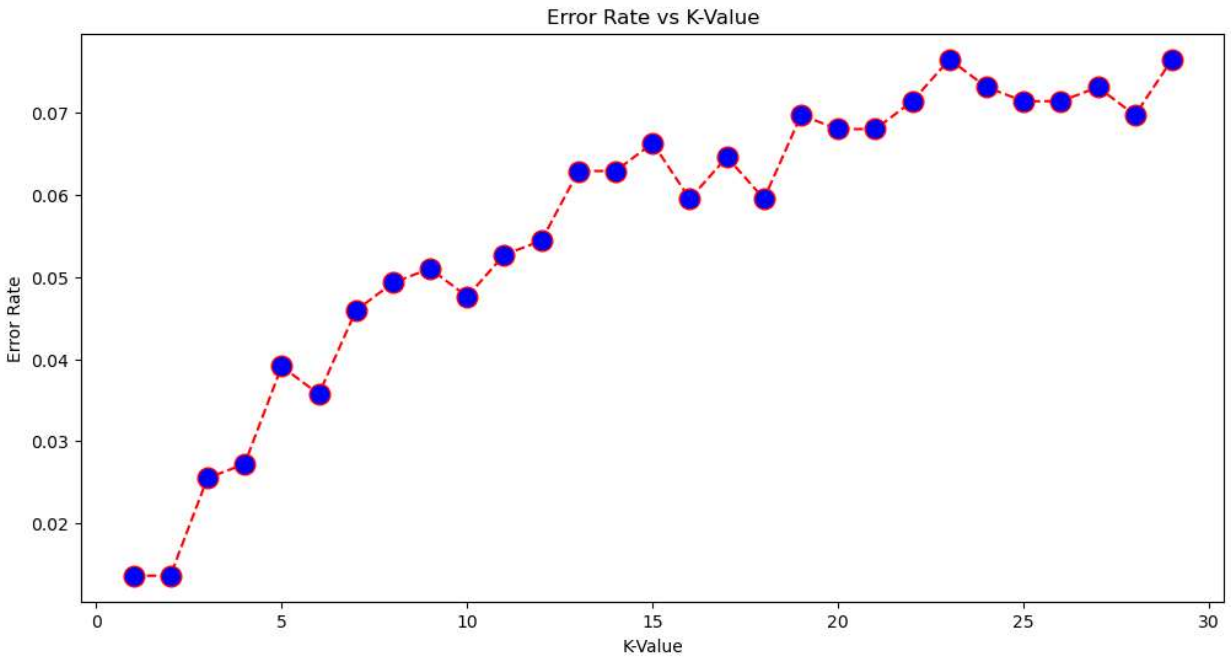
```
1.0

1.0
```

In [52]:
```
from sklearn.model_selection import cross_val_score
training_accuracy = cross_val_score(xgb, x_train, y_train, cv=10)
test_accuracy = cross_val_score(xgb, x_test, y_test, cv=10)
print("Train Accuracy", training_accuracy)
print()
print("Train Mean Accuracy", training_accuracy.mean())
print()
print("Train Max Accuracy", training_accuracy.max())
print()
print("Test Accuracy", test_accuracy)
print()
print("Test Mean Accuracy", test_accuracy.mean())
print()
print("Test Max Accuracy", test_accuracy.max())
train_mean.append(training_accuracy.mean())
test_mean.append(test_accuracy.mean())
```

```
Train Accuracy [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

Train Mean Accuracy 1.0

Train Max Accuracy 1.0

Test Accuracy [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

Test Mean Accuracy 1.0

Test Max Accuracy 1.0
```

In [53]:
```
from sklearn.neighbors import KNeighborsClassifier
error_rate = []

for i in range(1,30):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(x_train, y_train)
    y_pred = knn.predict(x_test)
    error_rate.append(np.mean(y_pred !=y_test))
```

In [54]:
```
plt.figure(figsize=(12,6))
plt.plot(range(1,30), error_rate, color='red', linestyle='dashed', marker='o',
        markersize=12, markerfacecolor='blue')
plt.title("Error Rate vs K-Value")
plt.xlabel("K-Value")
plt.ylabel("Error Rate")
plt.show()
```




In [55]:
```
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(x_train,y_train)
```

Out[55]:
```
KNeighborsClassifier(n_neighbors=1)
```

In [56]:
```
train_pred_knn = knn.predict(x_train)
test_pred_knn = knn.predict(x_test)
print(confusion_matrix(y_train,train_pred_knn))
print()
print(confusion_matrix(y_test,test_pred_knn))
```

```
[[881   0]
 [  0 881]]

[[286   8]
 [  0 294]]
```

In [57]:
```
print(classification_report(y_train,train_pred_knn))
print()
print(classification_report(y_test,test_pred_knn))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       881
           1       1.00      1.00      1.00       881

    accuracy                           1.00      1762
   macro avg       1.00      1.00      1.00      1762
weighted avg       1.00      1.00      1.00      1762


              precision    recall  f1-score   support

           0       1.00      0.97      0.99       294
           1       0.97      1.00      0.99       294

    accuracy                           0.99       588
   macro avg       0.99      0.99      0.99       588
weighted avg       0.99      0.99      0.99       588
```

In [58]:
```
print(accuracy_score(y_train,train_pred_knn))
print()
train_accu.append(accuracy_score(y_train,train_pred_knn))
print(accuracy_score(y_test,test_pred_knn))
test_accu.append(accuracy_score(y_test,test_pred_knn))
```

```
1.0

0.9863945578231292
```

In [59]:
```
from sklearn.model_selection import cross_val_score
training_accuracy = cross_val_score(knn, x_train, y_train, cv=10)
test_accuracy = cross_val_score(knn, x_test, y_test, cv=10)
print("Train Accuracy", training_accuracy)
print()
print("Train Mean Accuracy", training_accuracy.mean())
print()
print("Train Max Accuracy", training_accuracy.max())
print()
print("Test Accuracy", test_accuracy)
print()
print("Test Mean Accuracy", test_accuracy.mean())
print()
print("Test Max Accuracy", test_accuracy.max())
```

```
train_mean.append(training_accuracy.mean())
test_mean.append(test_accuracy.mean())
```

```
Train Accuracy [0.98870056 1.         0.96590909 0.98295455 0.98863636 0.98863636
 0.99431818 0.98863636 0.98863636 0.98295455]

Train Mean Accuracy 0.9869382383153569

Train Max Accuracy 1.0

Test Accuracy [0.91525424 0.96610169 0.91525424 0.93220339 0.93220339 0.98305085
 0.96610169 0.94915254 0.94827586 0.93103448]

Test Mean Accuracy 0.9438632378725892

Test Max Accuracy 0.9830508474576272
```

In [60]:
```
# Support Vector Machine classification algorithm
from sklearn.svm import SVC
svm = SVC(kernel = 'rbf')
svm.fit(x_train,y_train)
```

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

In [61]:
```
train_pred_svm = svm.predict(x_train)
test_pred_svm = svm.predict(x_test)
print(confusion_matrix(y_train,train_pred_svm))
print()
print(confusion_matrix(y_test,test_pred_svm))
```

```
[[881   0]
 [  0 881]]

[[294   0]
 [  0 294]]
```

In [62]:
```
print(classification_report(y_train,train_pred_svm))
print()
print(classification_report(y_test,test_pred_svm))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       881
           1       1.00      1.00      1.00       881

    accuracy                           1.00      1762
   macro avg       1.00      1.00      1.00      1762
weighted avg       1.00      1.00      1.00      1762


              precision    recall  f1-score   support

           0       1.00      1.00      1.00       294
           1       1.00      1.00      1.00       294

    accuracy                           1.00       588
   macro avg       1.00      1.00      1.00       588
weighted avg       1.00      1.00      1.00       588
```

In [63]:
```
print(accuracy_score(y_train,train_pred_svm))
train_accu.append(accuracy_score(y_train,train_pred_svm))
print()
print(accuracy_score(y_test,test_pred_svm))
test_accu.append(accuracy_score(y_test,test_pred_svm))
```

```
1.0

1.0
```

In [64]:
```
from sklearn.model_selection import cross_val_score
training_accuracy = cross_val_score(svm, x_train, y_train, cv=10)
test_accuracy = cross_val_score(svm, x_test, y_test, cv=10)
print("Train Accuracy", training_accuracy)
print()
print("Train Mean Accuracy", training_accuracy.mean())
print()
print("Train Max Accuracy", training_accuracy.max())
print()
print("Test Accuracy", test_accuracy)
print()
print("Test Mean Accuracy", test_accuracy.mean())
print()
print("Test Max Accuracy", test_accuracy.max())
train_mean.append(training_accuracy.mean())
test_mean.append(test_accuracy.mean())
```

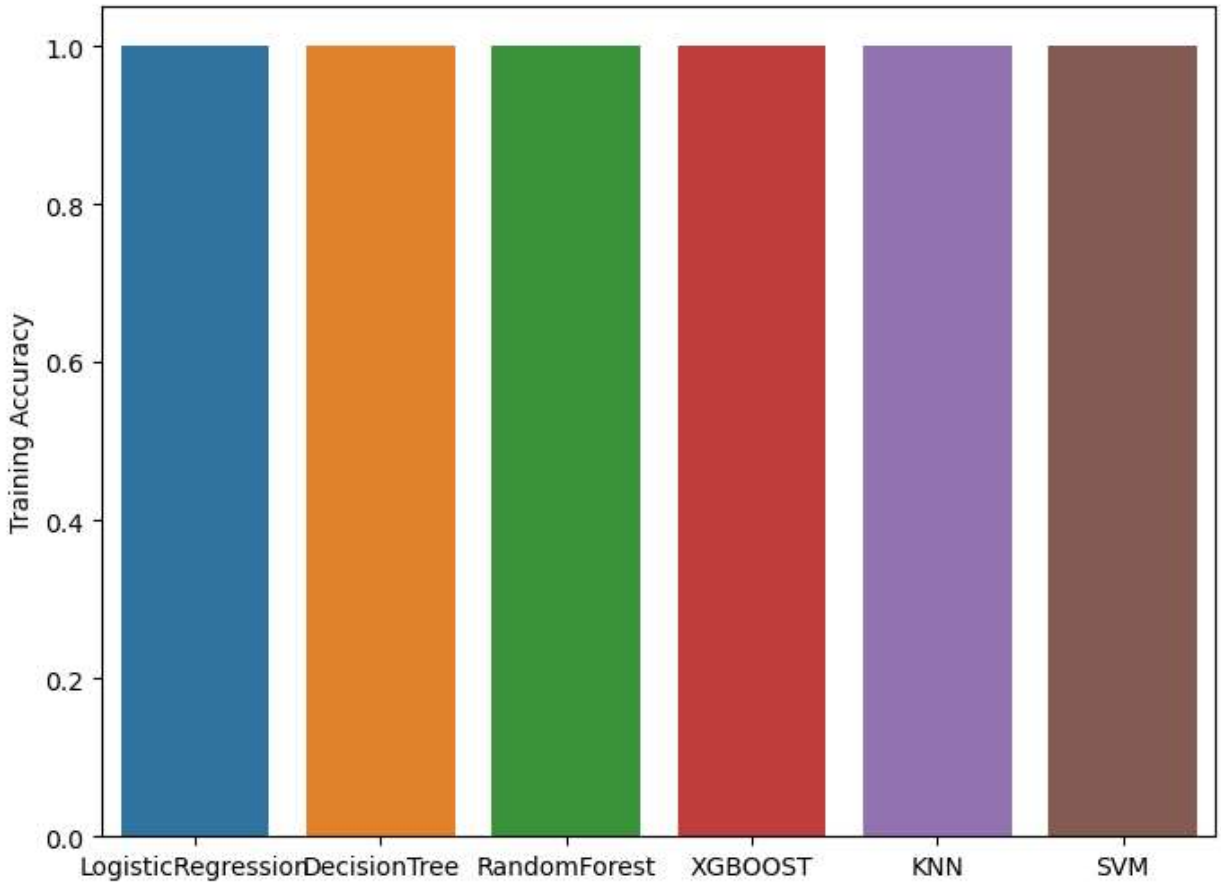
```
Train Accuracy [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

Train Mean Accuracy 1.0

Train Max Accuracy 1.0

Test Accuracy [0.98305085 1.         1.         1.         0.98305085 1.
 1.         0.98305085 0.98275862 1.        ]

Test Mean Accuracy 0.9931911163062537

Test Max Accuracy 1.0
```

In [65]:
```
algorithm = ['LogisticRegression','DecisionTree','RandomForest', 'XGBOOST','KNN','SVM'
accu_data = {'Training Accuracy':train_accu,'Test Accuracy':test_accu, 'Train Mean':tr
model = pd.DataFrame(accu_data, index = algorithm)
model
```

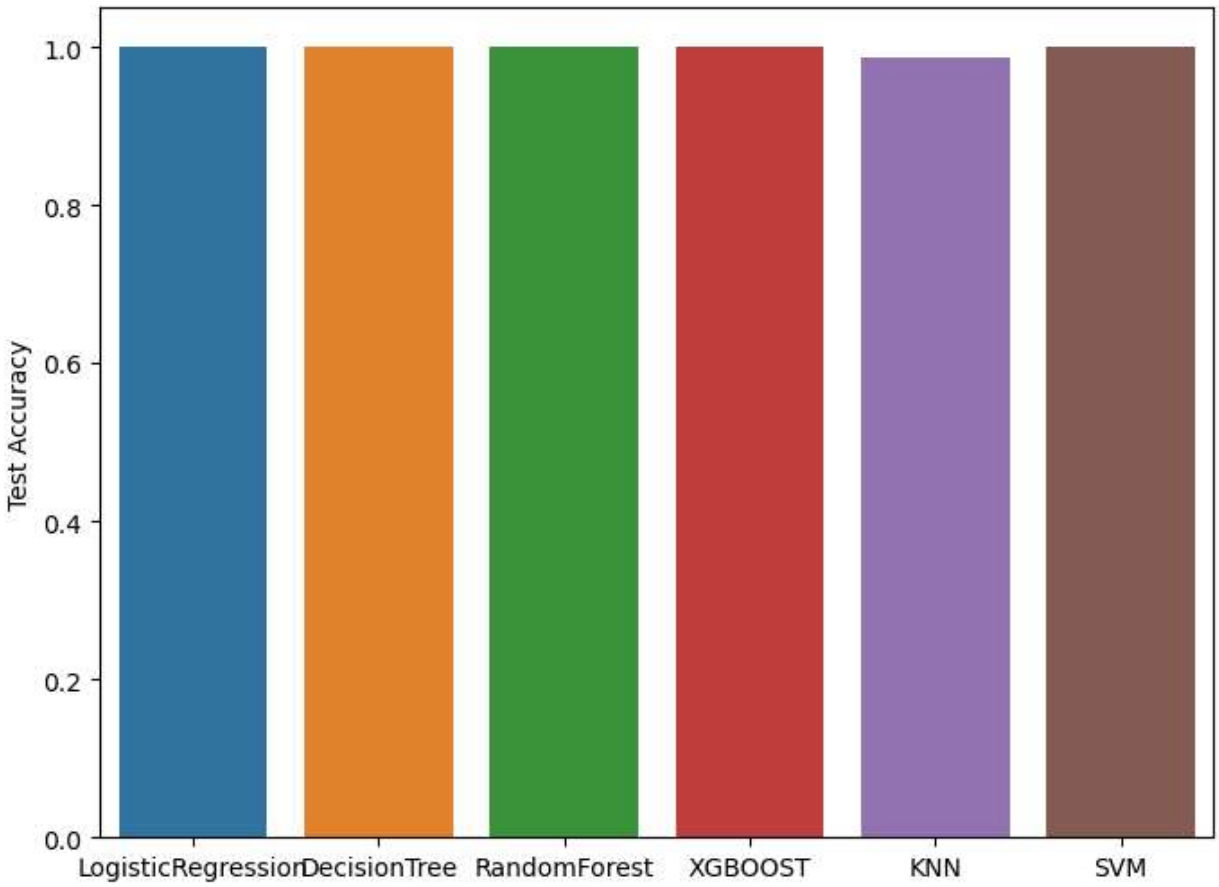
Out[65]:

| | Training Accuracy | Test Accuracy | Train Mean | Test Mean |
|---|---|---|---|---|
| **LogisticRegression** | 1.0 | 1.000000 | 1.000000 | 1.000000 |
| **DecisionTree** | 1.0 | 1.000000 | 1.000000 | 1.000000 |
| **RandomForest** | 1.0 | 1.000000 | 1.000000 | 1.000000 |
| **XGBOOST** | 1.0 | 1.000000 | 1.000000 | 1.000000 |
| **KNN** | 1.0 | 0.986395 | 0.986938 | 0.943863 |
| **SVM** | 1.0 | 1.000000 | 1.000000 | 0.993191 |

In [66]:
```
plt.figure(figsize=(8,6))
sns.barplot(x=model.index, y=model['Training Accuracy'])
plt.show()
```




In [67]:
```
plt.figure(figsize=(8,6))
sns.barplot(x=model.index, y=model['Test Accuracy'])
plt.show()
```


1.0
0.8
0.6
0.4
0.2
0.0
Test Accuracy
LogisticRegression
DecisionTree
RandomForest
XGBOOST
KNN
SVM

In [ ]: