In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
from scipy import stats
import statsmodels.api as sm
from statsmodels.formula.api import ols
from statsmodels.stats.multicomp import pairwise_tukeyhsd
from scipy.stats import chi2, chi2_contingency
import statsmodels.formula.api as smf
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
pd.set_option('display.max_column',None)
warnings.filterwarnings('ignore')
import re
```

In [39]:
```
red_wine = pd.read_csv('winequality-red.csv')
```

In [40]:
```
red_wine.shape
```

Out[40]:
```
(1599, 12)
```

In [41]:
```
red_wine.head()
```

Out[41]:

	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

In [42]:
```
red_wine.select_dtypes('number').columns.shape # all are continous variable
```

Out[42]:
```
(12,)
```

In [43]:
```
red_wine.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 [44]:
```
red_wine.describe()
```

Out[44]:

	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

1) quality(Target attribute)

In [45]:
```
bins = (2, 6.5, 8)
labels = ['bad', 'good']
red_wine['quality'] = pd.cut(x = red_wine['quality'], bins = bins, labels = labels
```

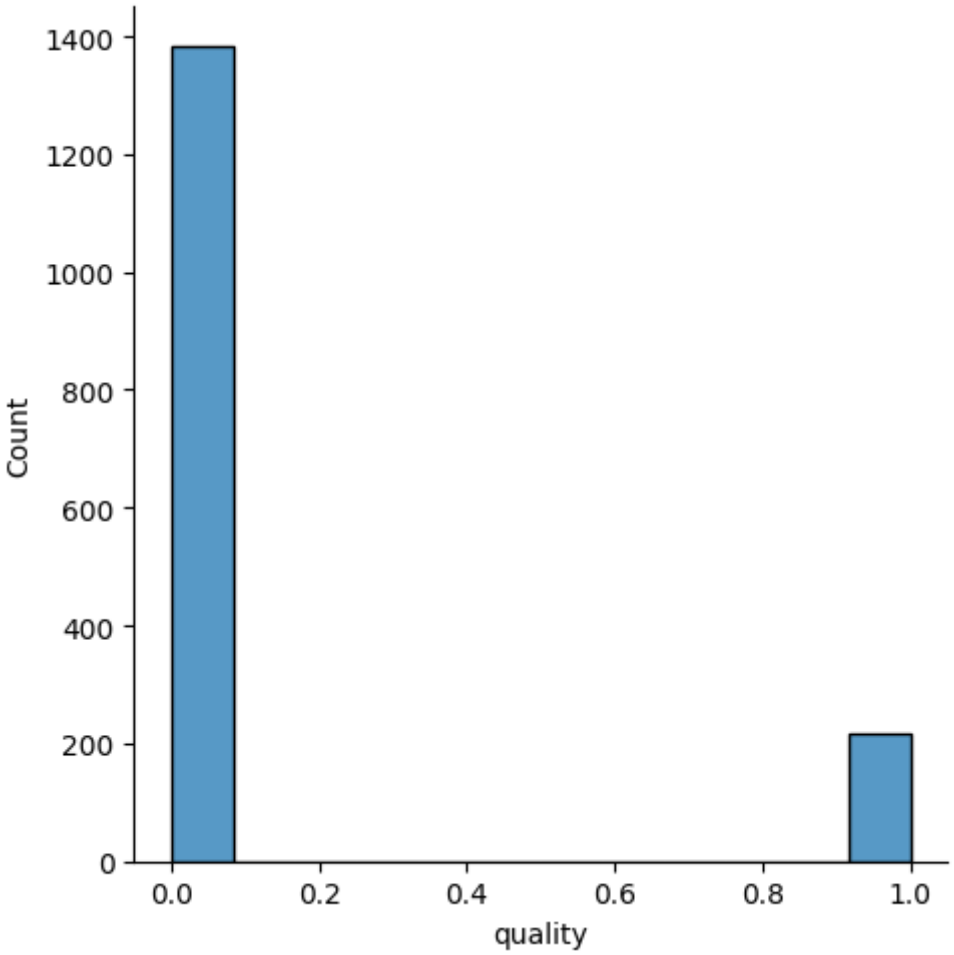
In [46]:
```
red_wine['quality'].value_counts() # here i'm considering this variable as continou
```

Out[46]:
```
bad     1382
good     217
Name: quality, dtype: int64
```

In [48]:
```
# giving intiger value
from sklearn.preprocessing import LabelEncoder
labelencoder_y = LabelEncoder()
red_wine['quality'] = labelencoder_y.fit_transform(red_wine['quality'])
```

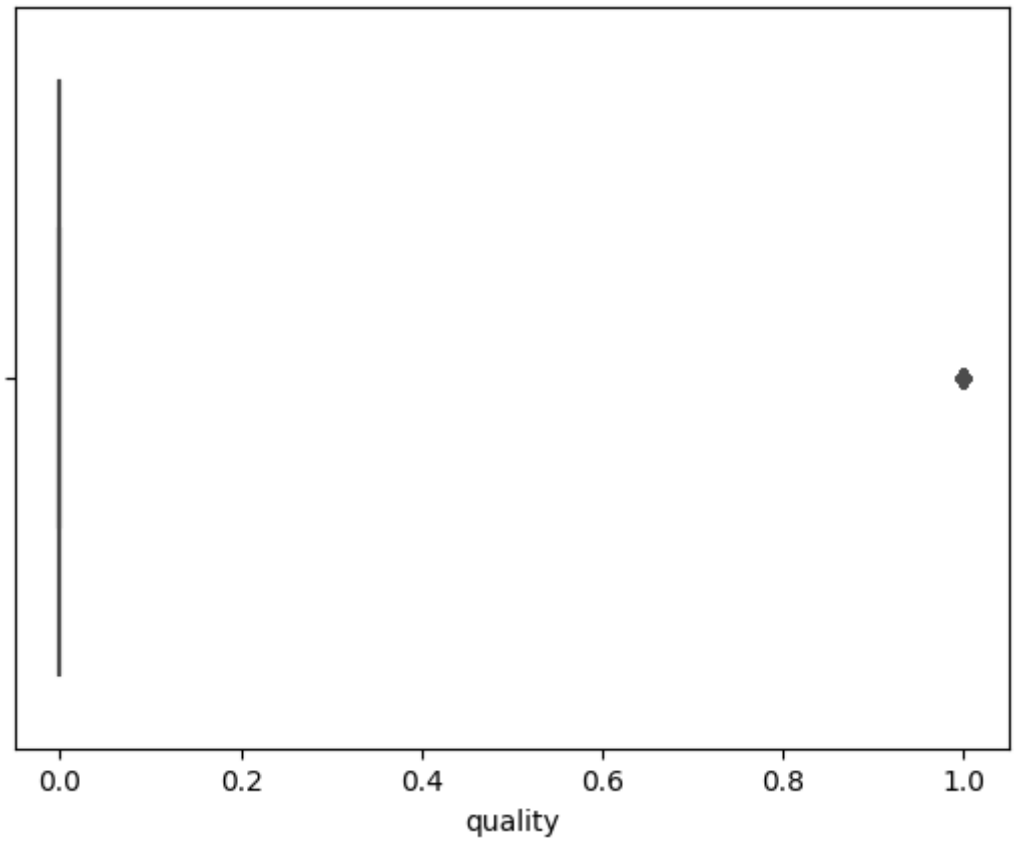
In [49]:
```
sns.displot(data = red_wine , x = 'quality' , kind='hist') # distribution of the "
```

Out[49]:
```
<seaborn.axisgrid.FacetGrid at 0x1868d8e9660>
```

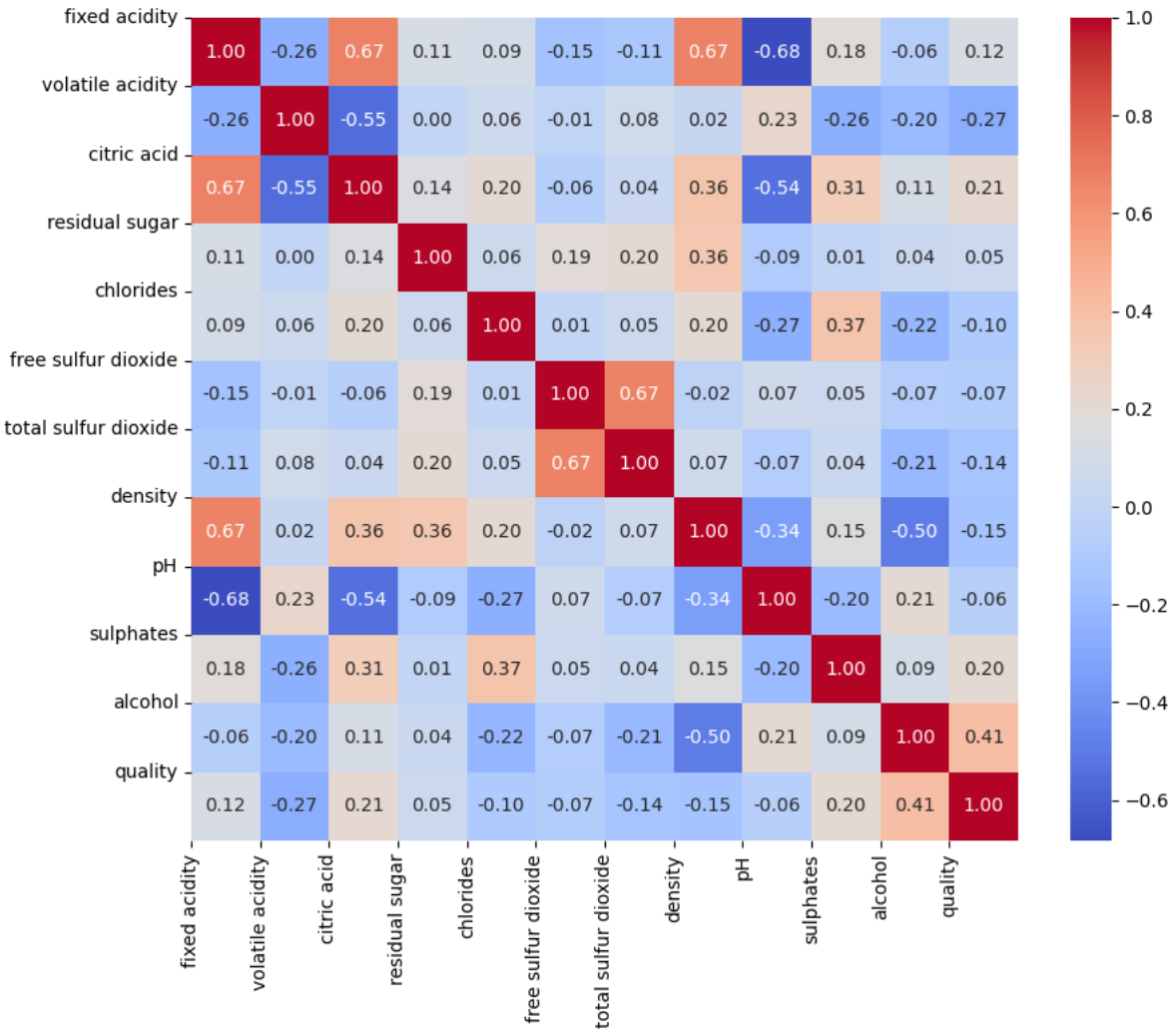


In [50]:
```
sns.boxplot(data = red_wine , x = 'quality' , color='r')
```

Out[50]:
```
<AxesSubplot: xlabel='quality'>
```



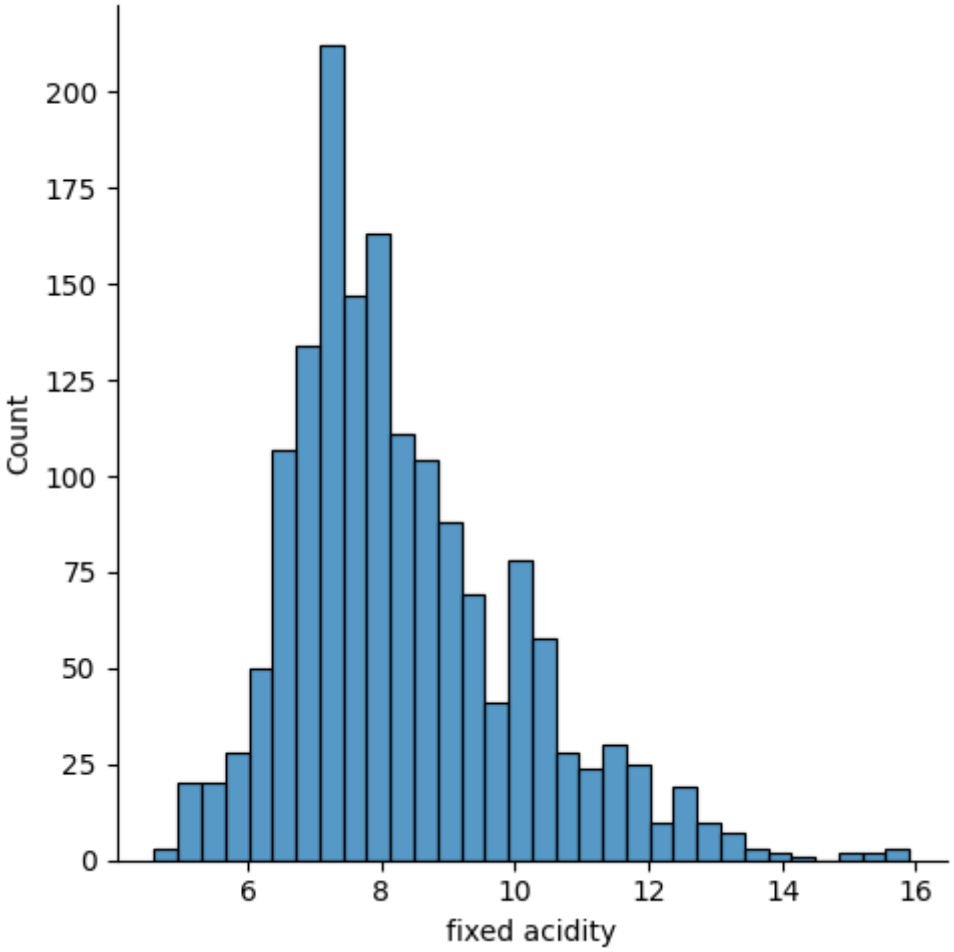
In [54]:
```
## Correlation matrix
corr = red_wine.corr()
#Plot figsize
fig, ax = plt.subplots(figsize=(10, 8))
#Generate Heat Map, allow annotations and place floats in map
sns.heatmap(corr, cmap='coolwarm', annot=True, fmt=".2f")
#Apply xticks
plt.xticks(range(len(corr.columns)), corr.columns);
#Apply yticks
plt.yticks(range(len(corr.columns)), corr.columns)
#show plot
plt.show()
```



2) fixed acidity

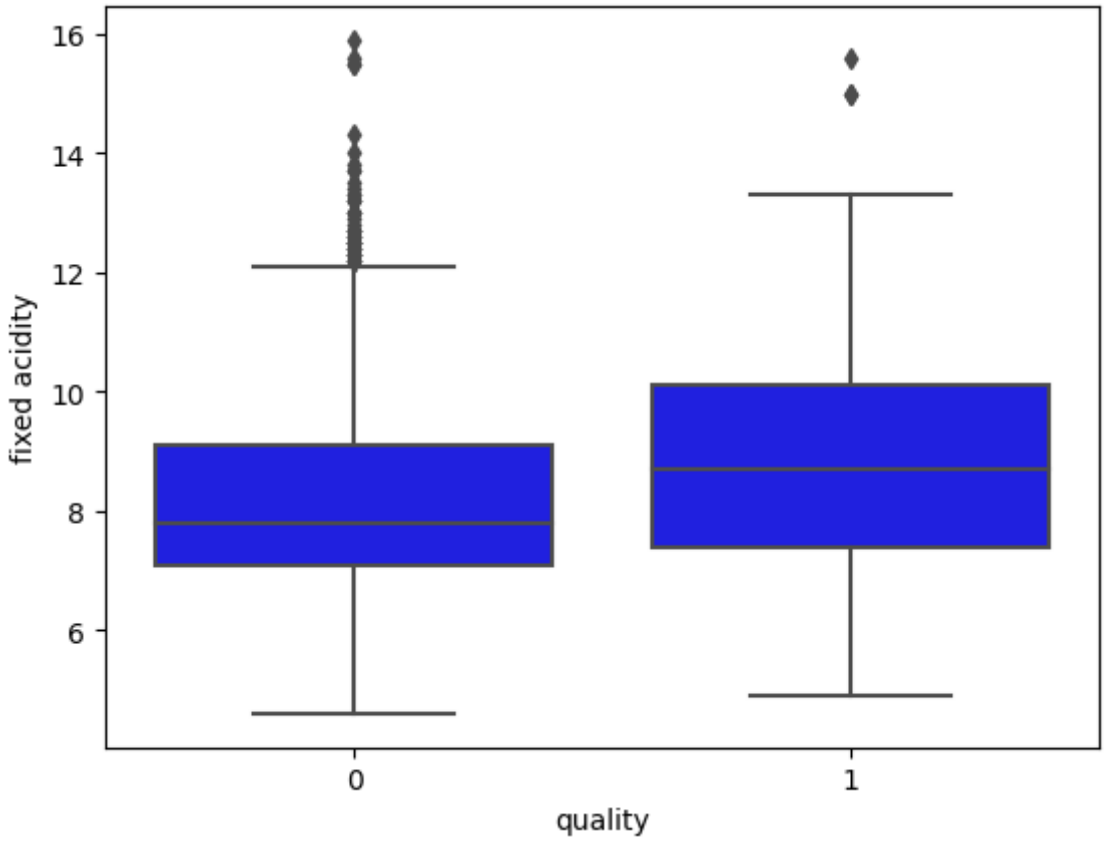
In [51]:
```
sns.displot(data = red_wine , x = 'fixed acidity' , kind='hist') # slightly positi
```

Out[51]:
```
<seaborn.axisgrid.FacetGrid at 0x1868d8ebdc0>
```



In [57]:
```
sns.boxplot(data = red_wine , x = 'quality' , y = 'fixed acidity' , color='b')
```

Out[57]:
```
<AxesSubplot: xlabel='quality', ylabel='fixed acidity'>
```



In [15]:
```
# fix the outliers
#iqr_fa = stats.iqr(red_wine['fixed acidity'])
#iqr_fa
```

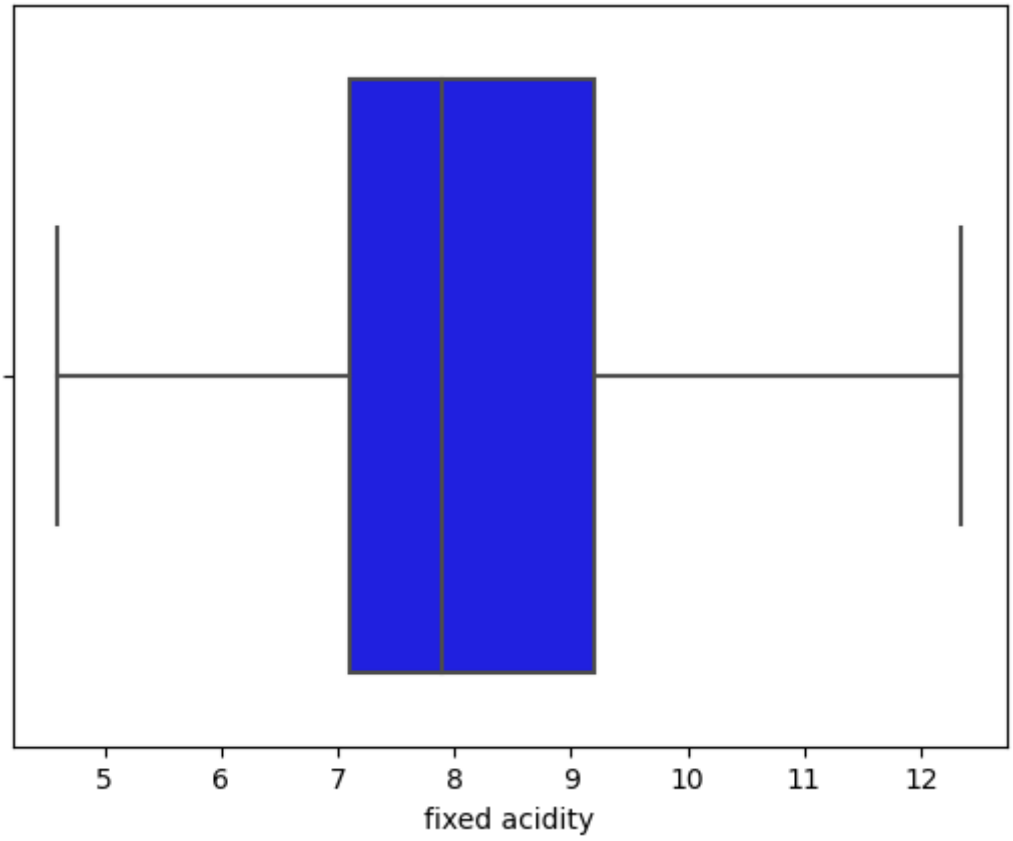
Out[15]:
```
2.0999999999999996
```

In [16]:
```
#Q1 = red_wine['fixed acidity'].quantile(0.25)

#Q3 = red_wine['fixed acidity'].quantile(0.75)

#upper_bound = Q3 + 1.5*iqr_fa
#lower_bound = Q1 - 1.5*iqr_fa
```

In [17]:
```
#red_wine['fixed acidity'] = np.where(red_wine['fixed acidity'] > upper_bound , upp
```

In [18]:
```
#sns.boxplot(data = red_wine , x = 'fixed acidity' , color='b') # outlier fixed
```

Out[18]:
```
<AxesSubplot: xlabel='fixed acidity'>
```



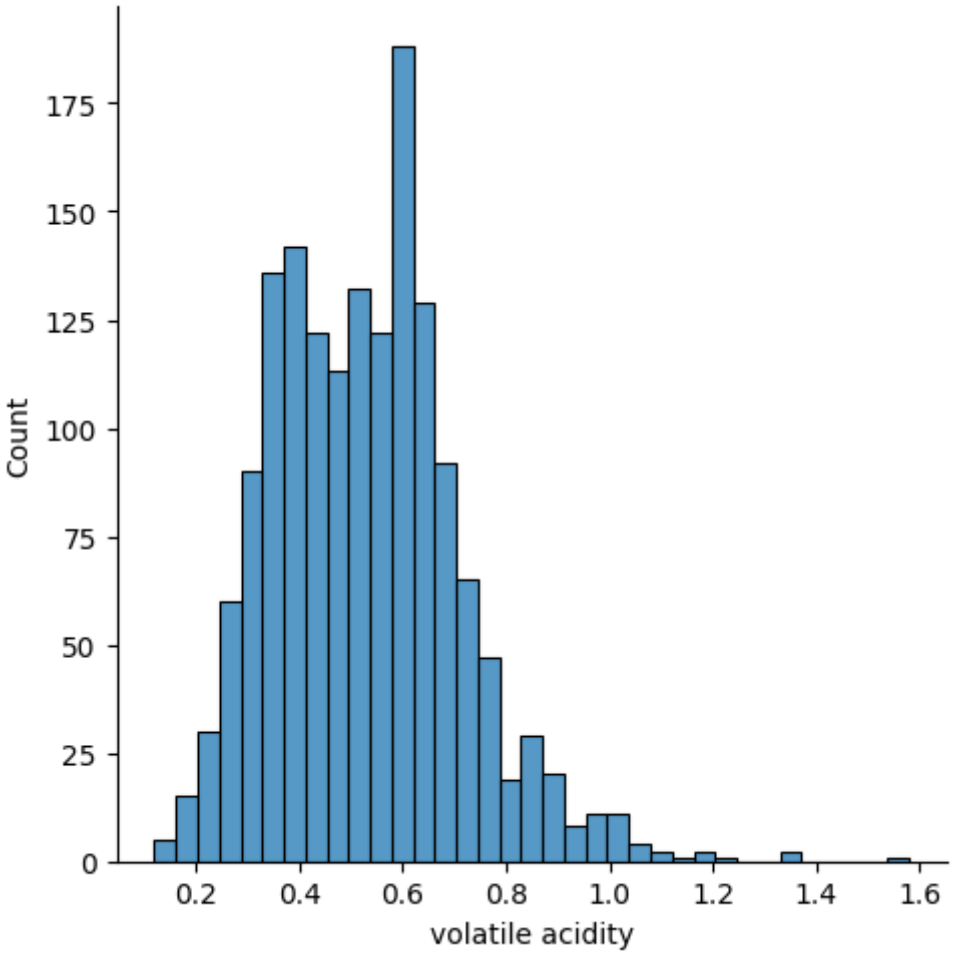
3) volatile acidity

In [19]:
```
red_wine['volatile acidity'].dtype
```

Out[19]:
```
dtype('float64')
```

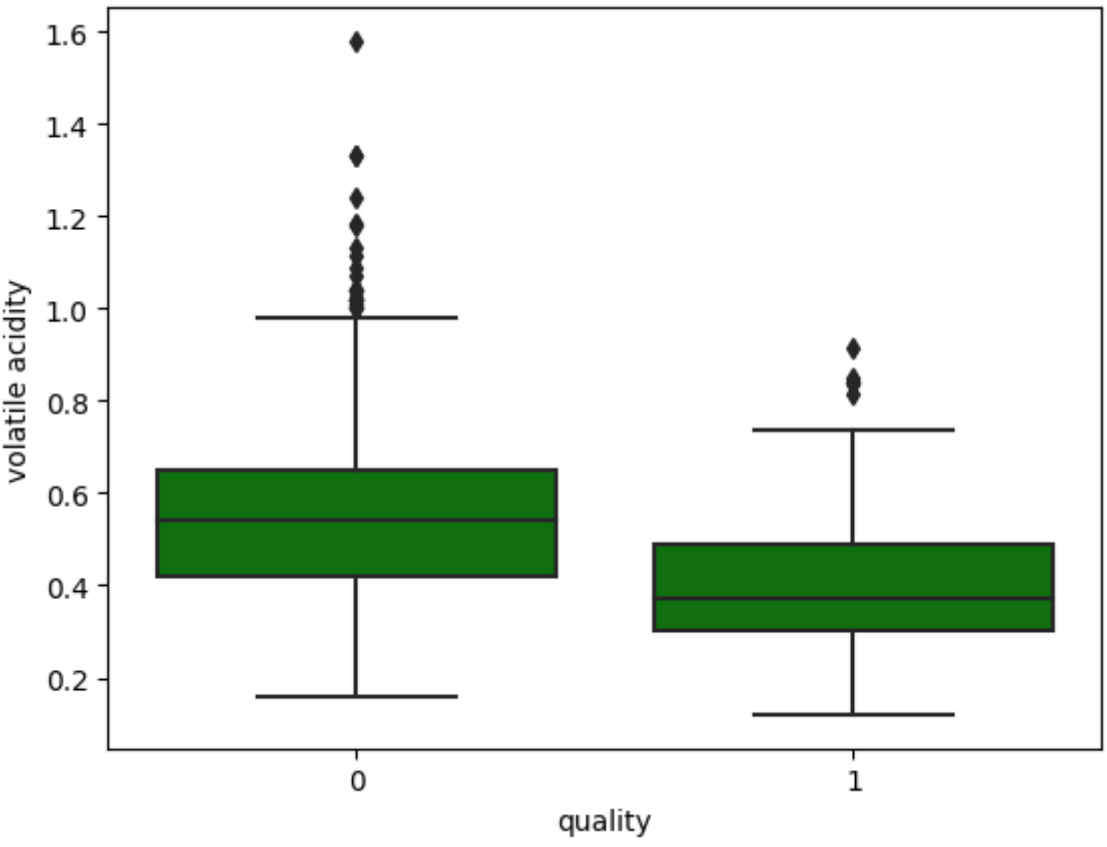
In [20]:
```
sns.displot(data = red_wine , x = 'volatile acidity') # positively skewed data
```

Out[20]:
```
<seaborn.axisgrid.FacetGrid at 0x186891c9b40>
```



In [58]:
```
# box plot
sns.boxplot(data = red_wine , x = 'quality' , y = 'volatile acidity' , color= 'g')
```

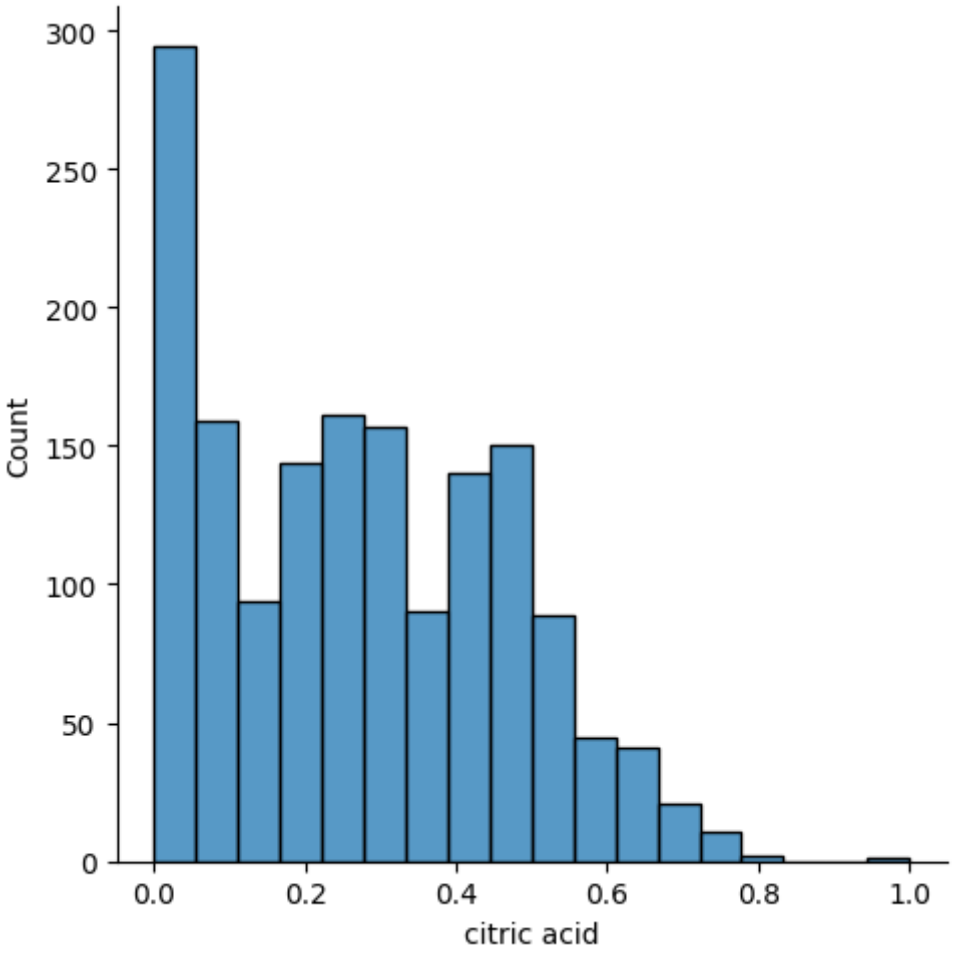
Out[58]:
```
<AxesSubplot: xlabel='quality', ylabel='volatile acidity'>
```



4) citric acid

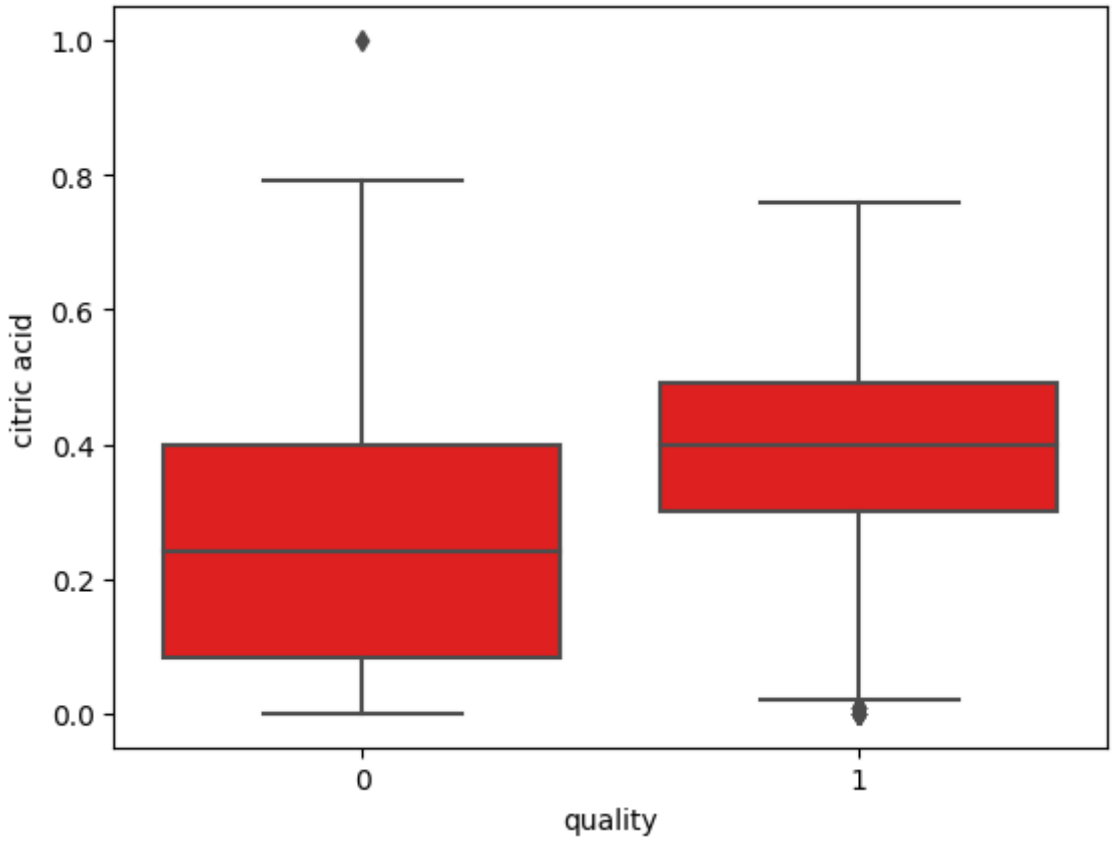
In [59]:
```
sns.displot(data = red_wine , x = 'citric acid')
```

Out[59]:
```
<seaborn.axisgrid.FacetGrid at 0x1868db71ba0>
```



In [60]:
```
sns.boxplot(data = red_wine , x = 'quality' , y = 'citric acid' , color= 'r')
```

Out[60]:
```
<AxesSubplot: xlabel='quality', ylabel='citric acid'>
```



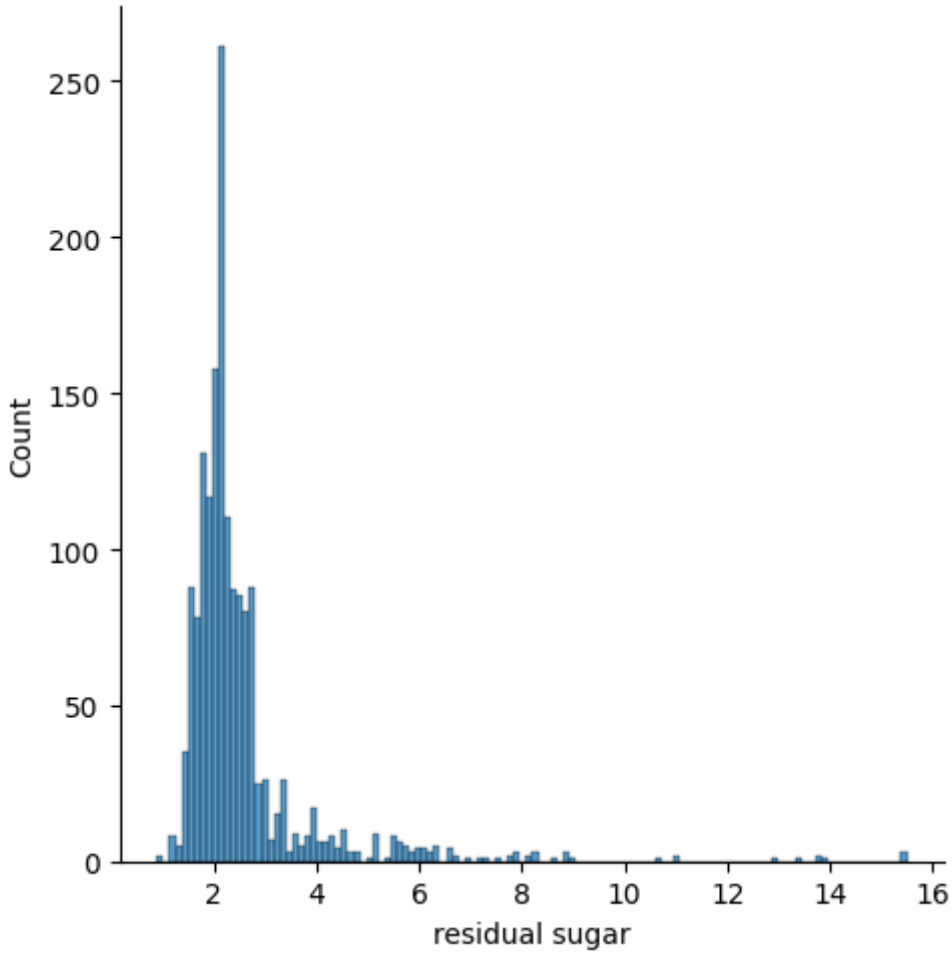
5) residual sugar

In [26]:
```
red_wine['residual sugar'].dtype
```

Out[26]:
```
dtype('float64')
```

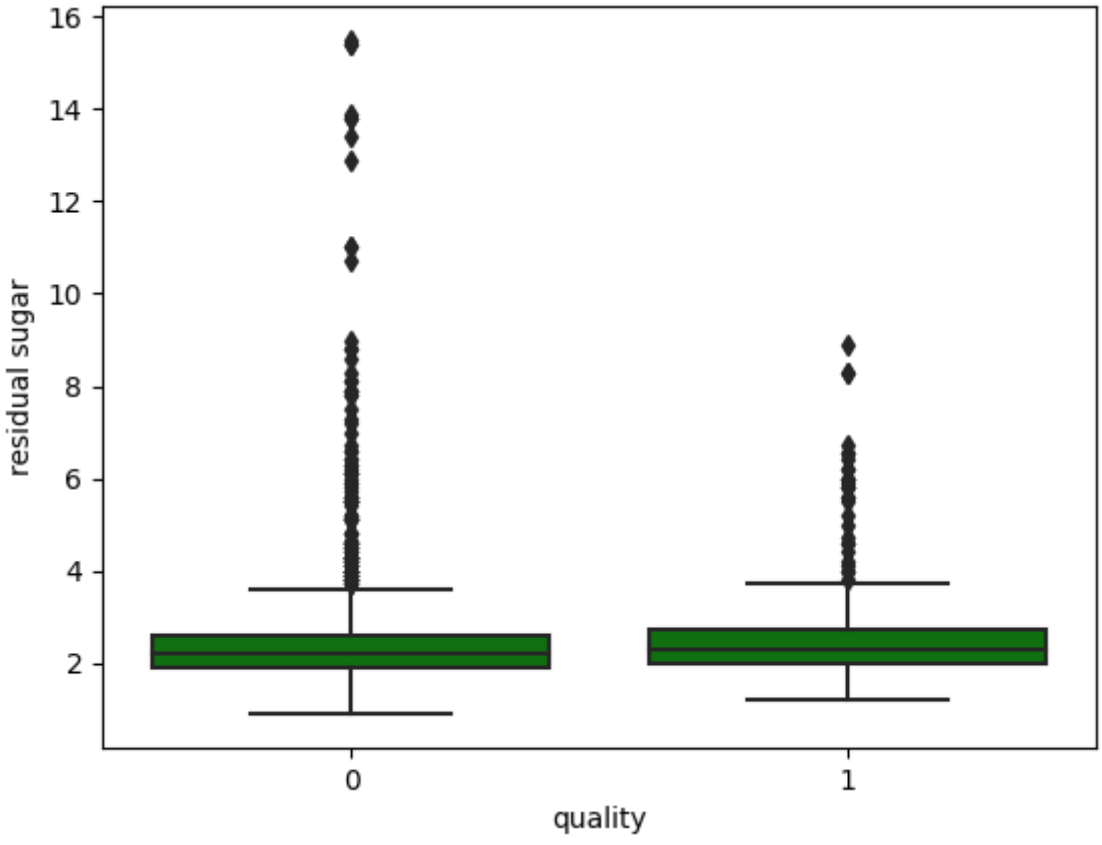
In [27]:
```
sns.displot(data = red_wine , x = 'residual sugar') # Positively skewed
```

Out[27]:
```
<seaborn.axisgrid.FacetGrid at 0x1868d6276a0>
```



In [61]:
```
sns.boxplot(data = red_wine , x = 'quality' , y = 'residual sugar' , color='g')
```

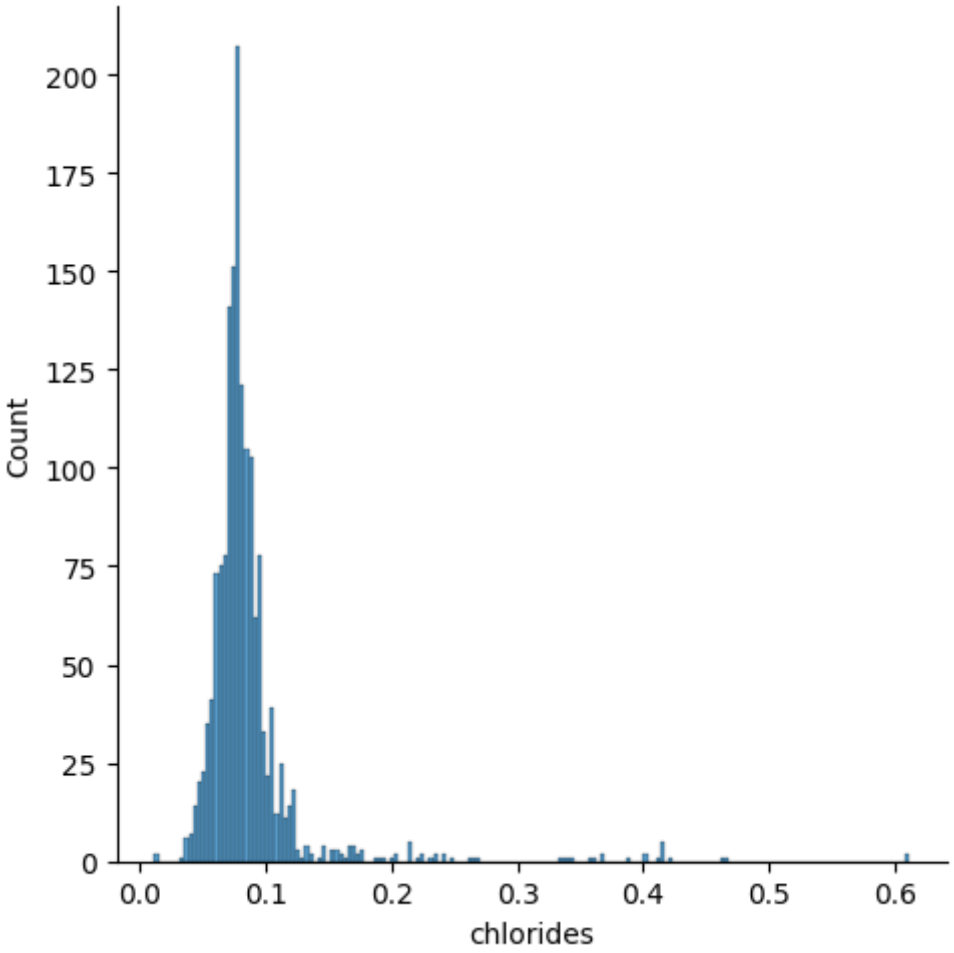
Out[61]:
```
<AxesSubplot: xlabel='quality', ylabel='residual sugar'>
```



6) chlorides

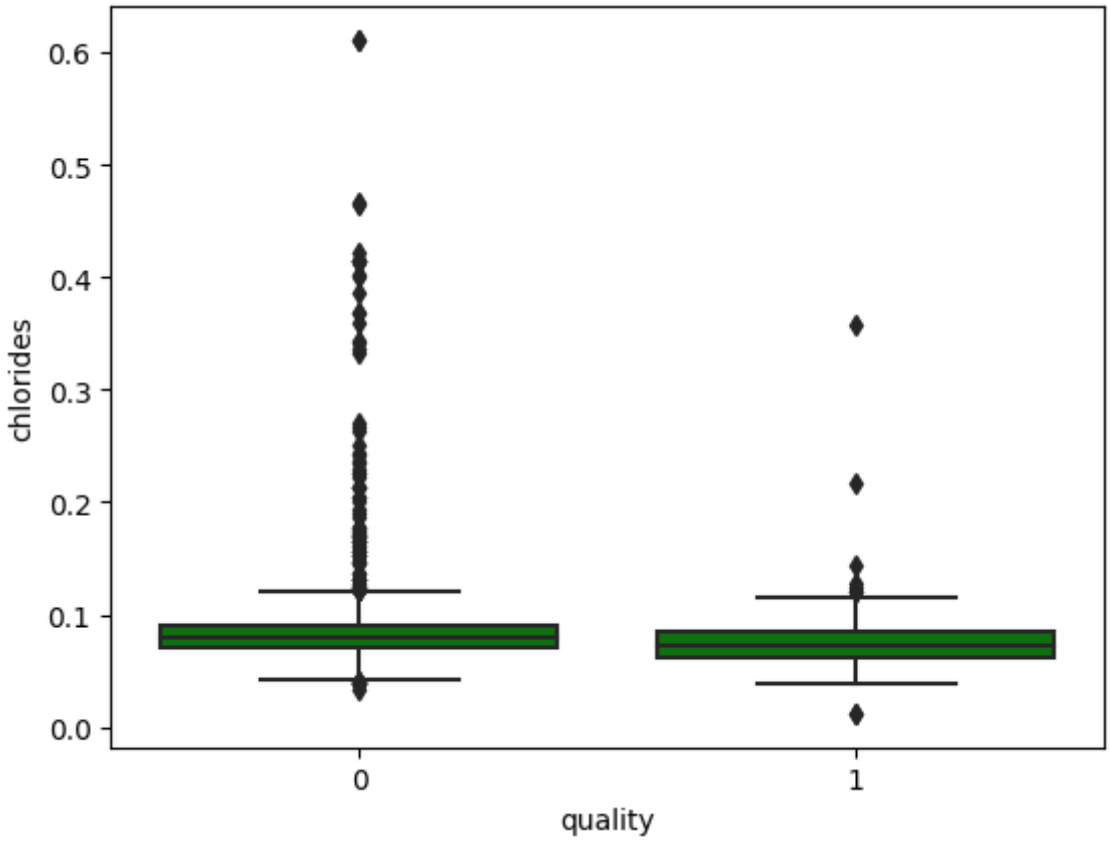
In [62]:
```
sns.displot(data = red_wine , x = 'chlorides')
```

Out[62]:
```
<seaborn.axisgrid.FacetGrid at 0x1868d5b3c40>
```



In [63]:
```
sns.boxplot(data = red_wine , x = 'quality' , y = 'chlorides' , color='g')
```

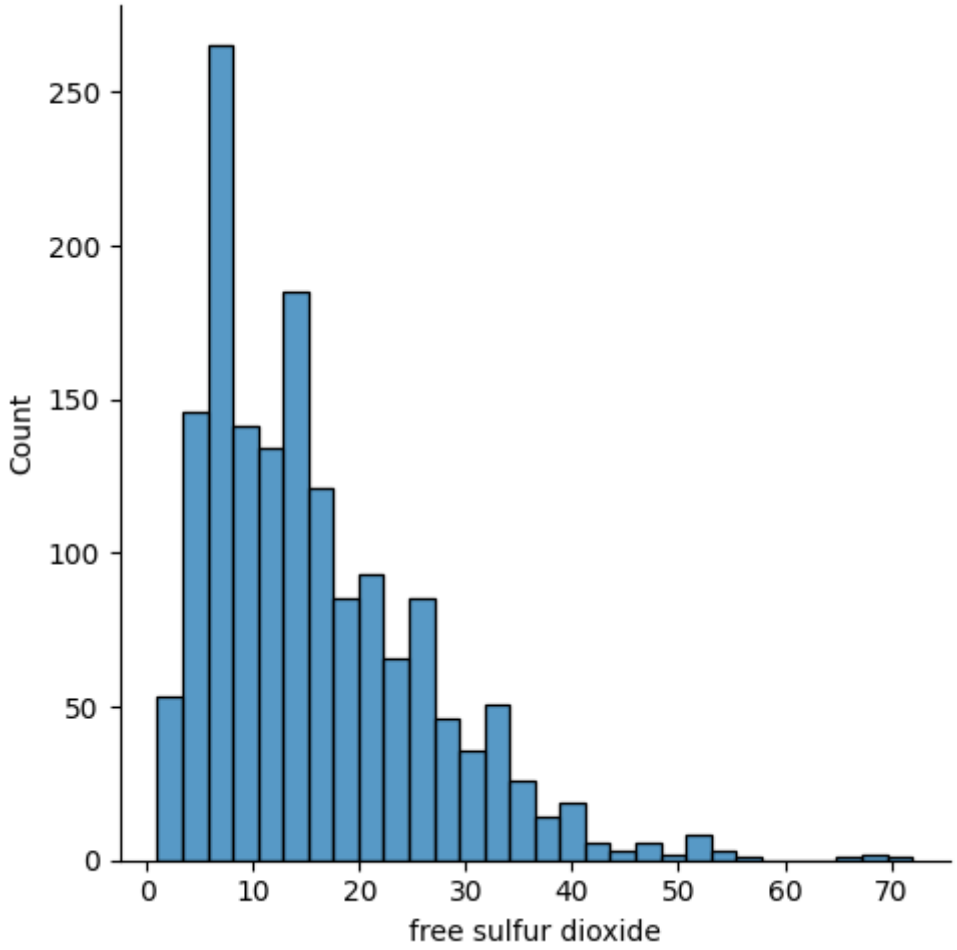
Out[63]:
```
<AxesSubplot: xlabel='quality', ylabel='chlorides'>
```



7) free sulfur dioxide

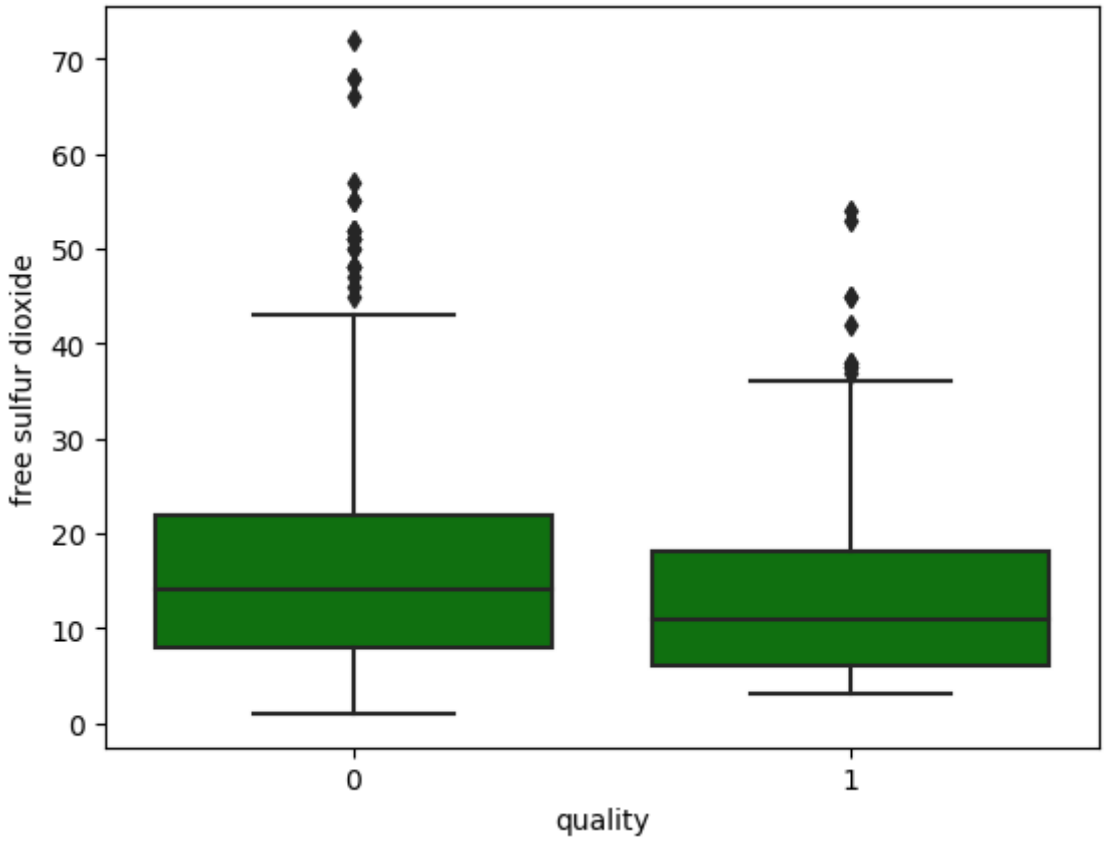
In [64]:
```
sns.displot(data = red_wine , x = 'free sulfur dioxide')
```

Out[64]:
```
<seaborn.axisgrid.FacetGrid at 0x1868936abc0>
```



In [65]:
```
sns.boxplot(data = red_wine , x = 'quality' , y = 'free sulfur dioxide' , color='g
```

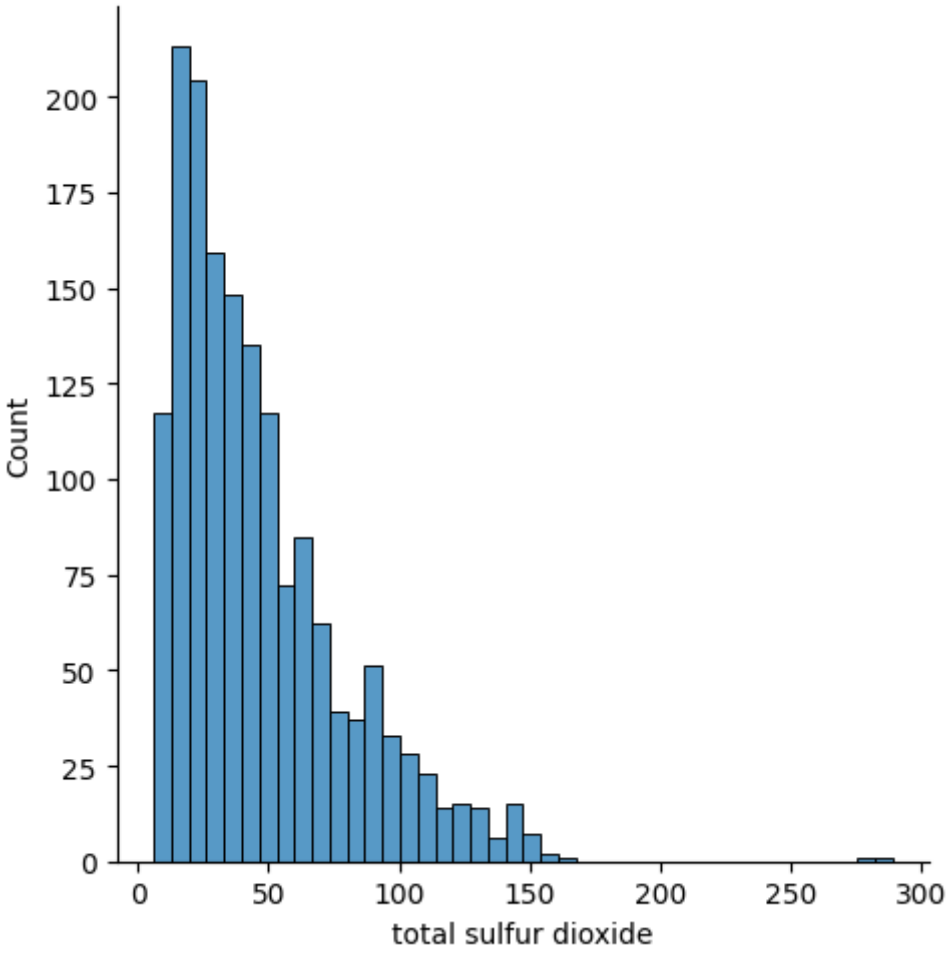
Out[65]:
```
<AxesSubplot: xlabel='quality', ylabel='free sulfur dioxide'>
```



8) total sulfur dioxide

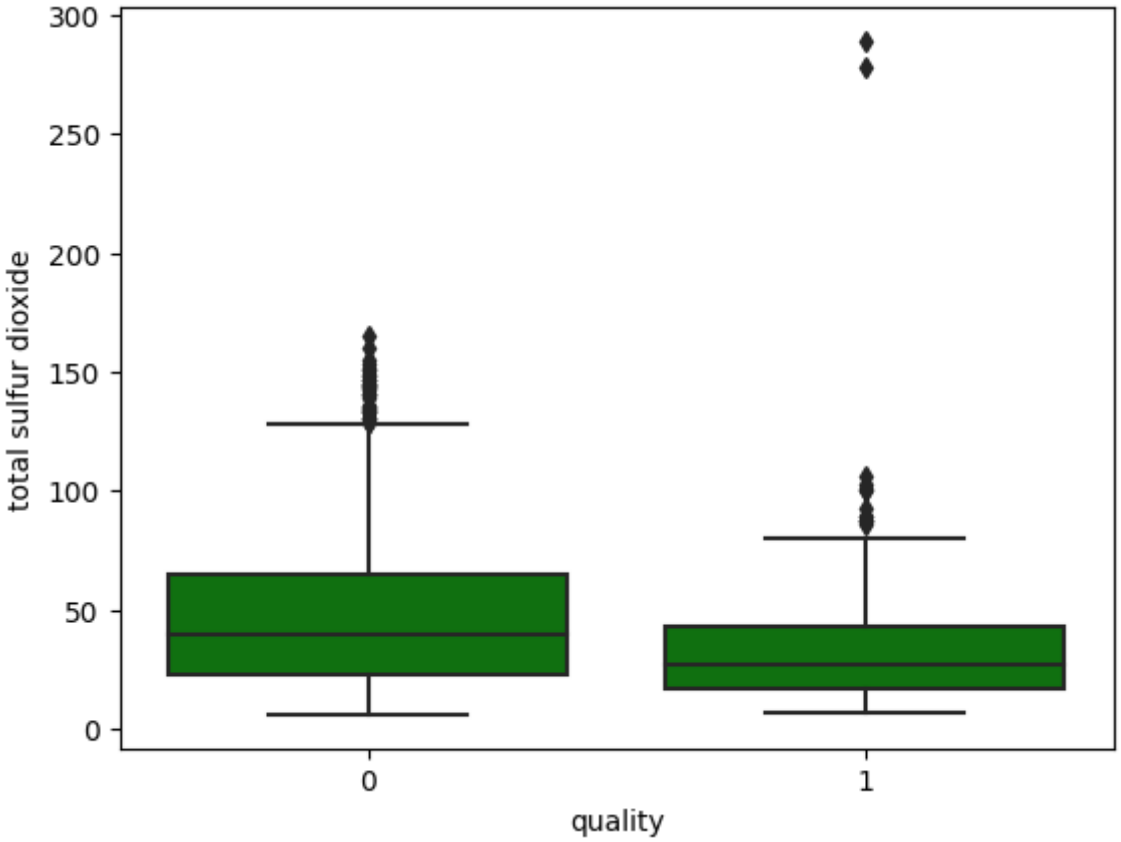
In [66]:
```
sns.displot(data = red_wine , x = 'total sulfur dioxide')
```

Out[66]:
```
<seaborn.axisgrid.FacetGrid at 0x1868cf66890>
```



In [67]:
```
sns.boxplot(data = red_wine , x = 'quality' , y = 'total sulfur dioxide' , color='
```

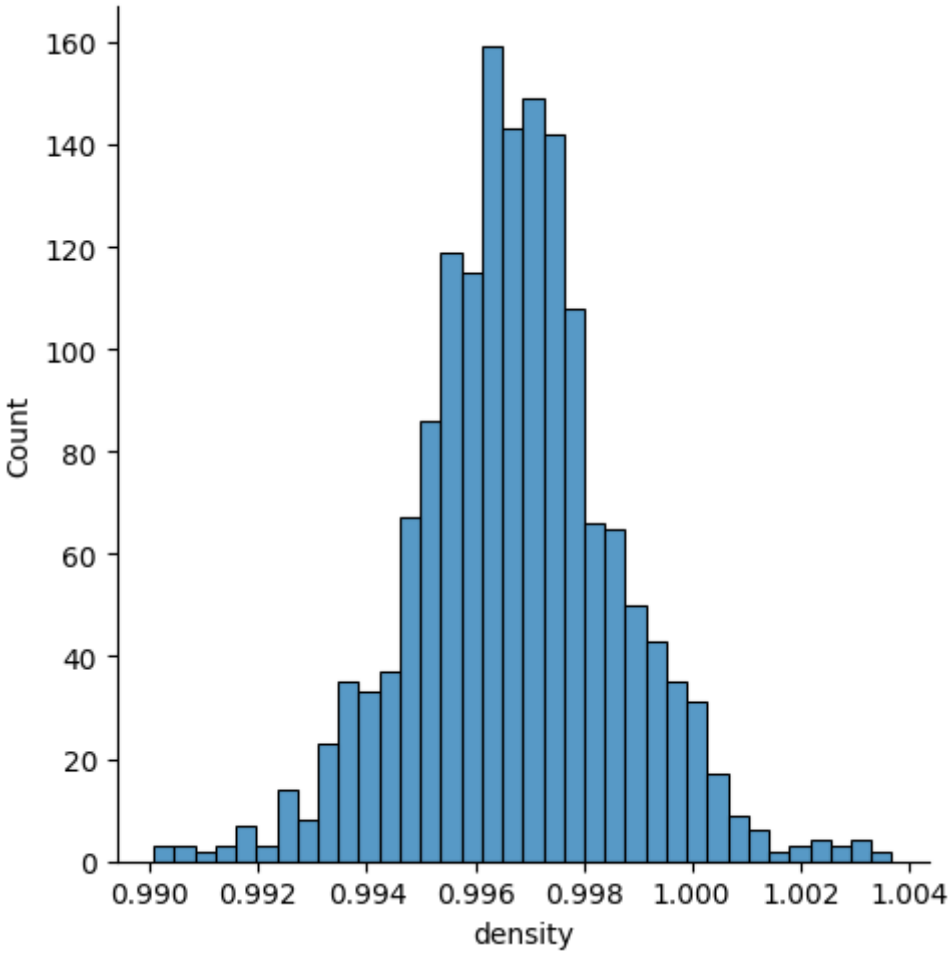
Out[67]:
```
<AxesSubplot: xlabel='quality', ylabel='total sulfur dioxide'>
```



9) density

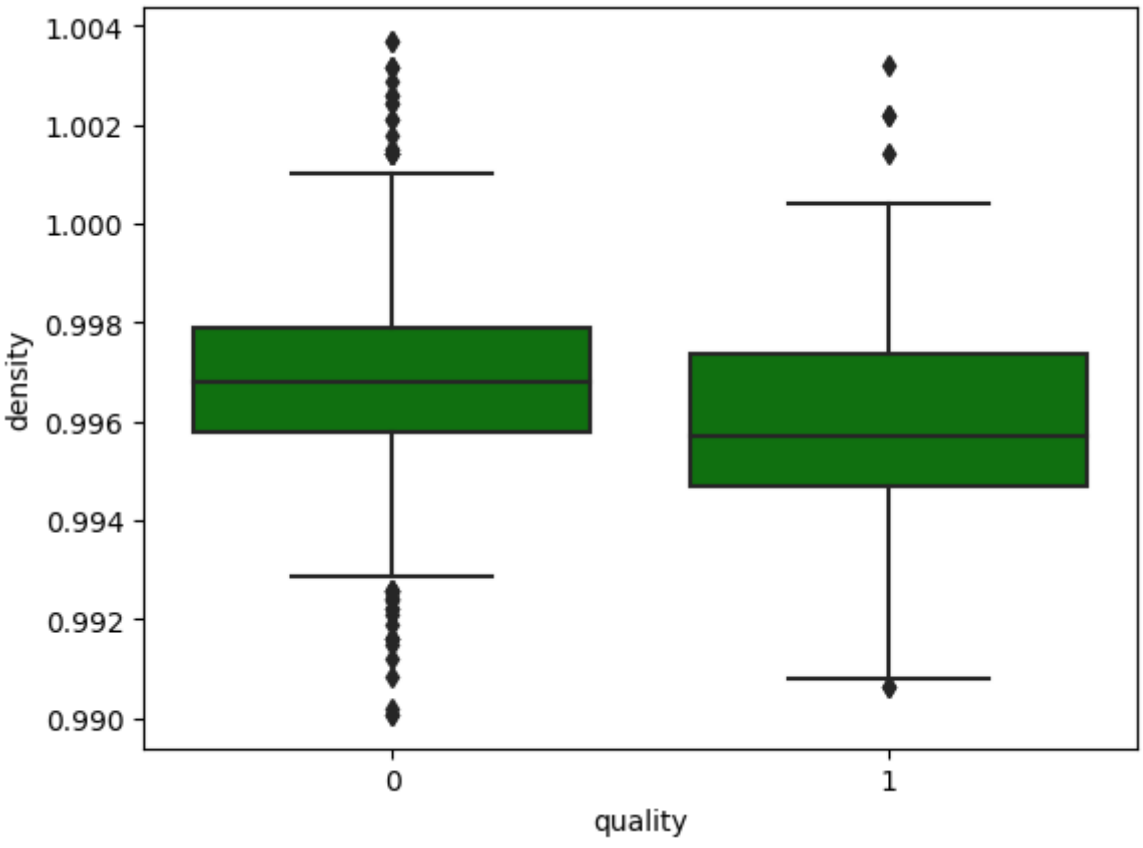
In [68]:
```
sns.displot(data = red_wine , x = 'density')
```

Out[68]:
```
<seaborn.axisgrid.FacetGrid at 0x1868c6847c0>
```



In [69]:
```
sns.boxplot(data = red_wine , x = 'quality' , y = 'density' , color='g')
```

Out[69]:
```
<AxesSubplot: xlabel='quality', ylabel='density'>
```



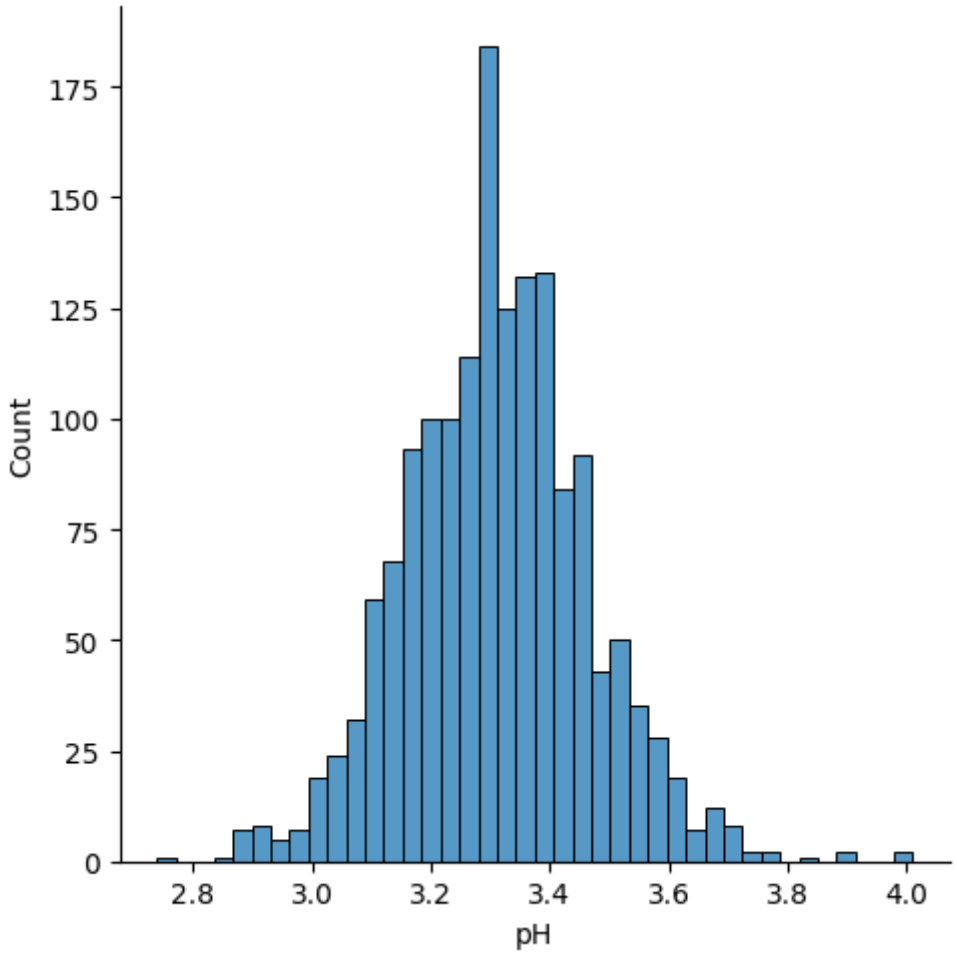
10) pH

In [70]:
```
red_wine.pH.value_counts()
```

Out[70]:
```
3.30    57
3.36    56
3.26    53
3.38    48
3.39    48
        ..
3.75     1
2.74     1
3.70     1
3.85     1
2.90     1
Name: pH, Length: 89, dtype: int64
```

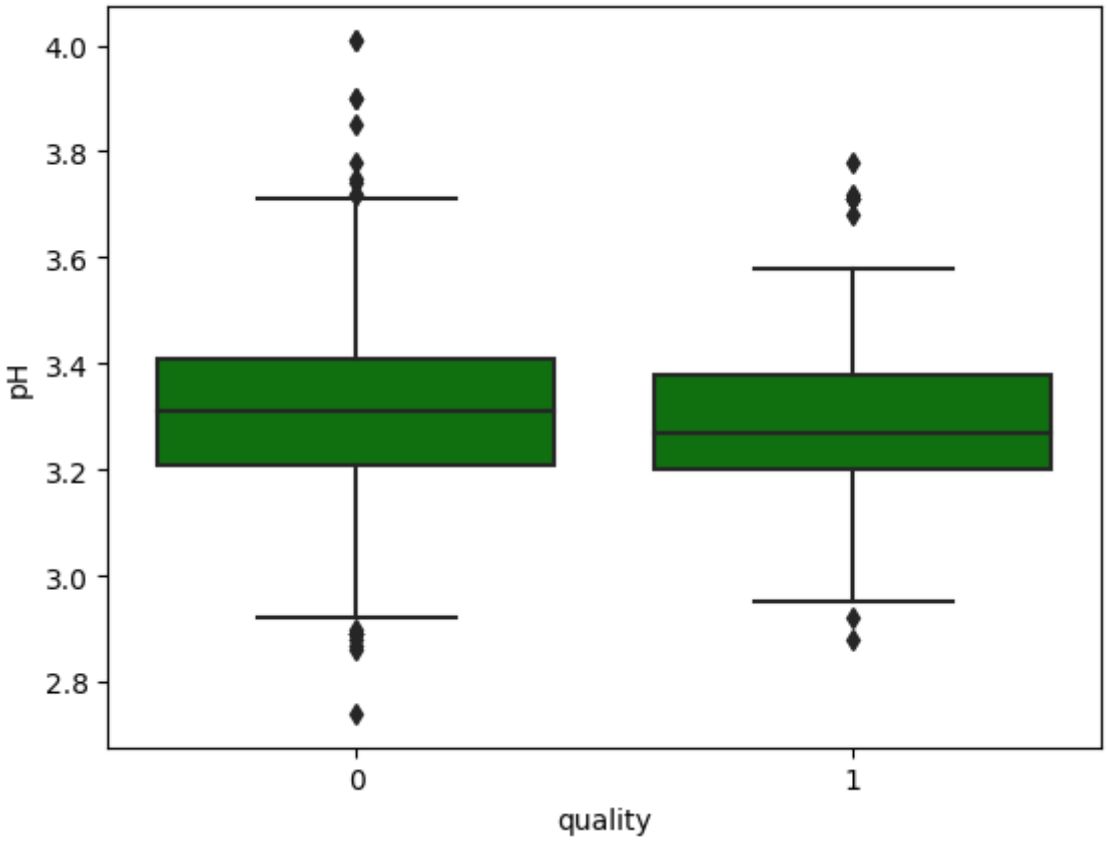
In [71]:
```
sns.displot(data = red_wine , x = 'pH')
```

Out[71]:
```
<seaborn.axisgrid.FacetGrid at 0x18692ab0bb0>
```



In [72]:
```
sns.boxplot(data = red_wine , x = 'quality' , y = 'pH' , color='g')
```

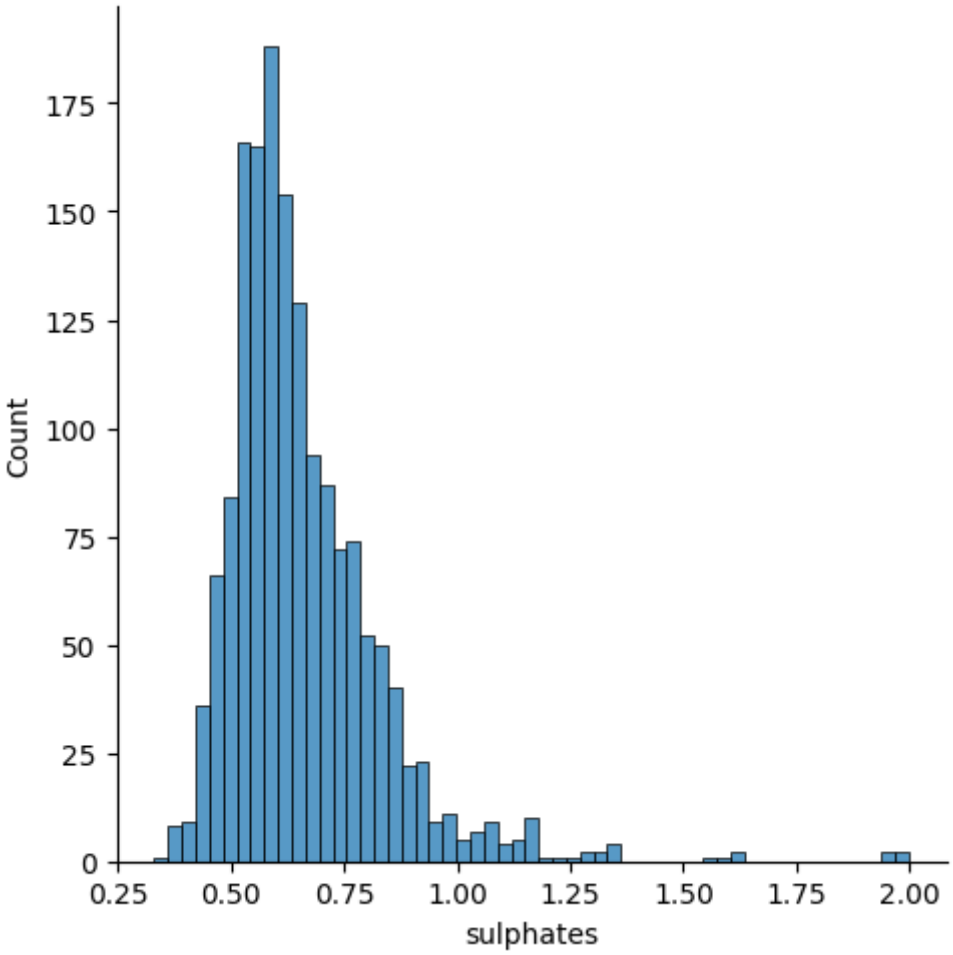
Out[72]:
```
<AxesSubplot: xlabel='quality', ylabel='pH'>
```



11) sulphates

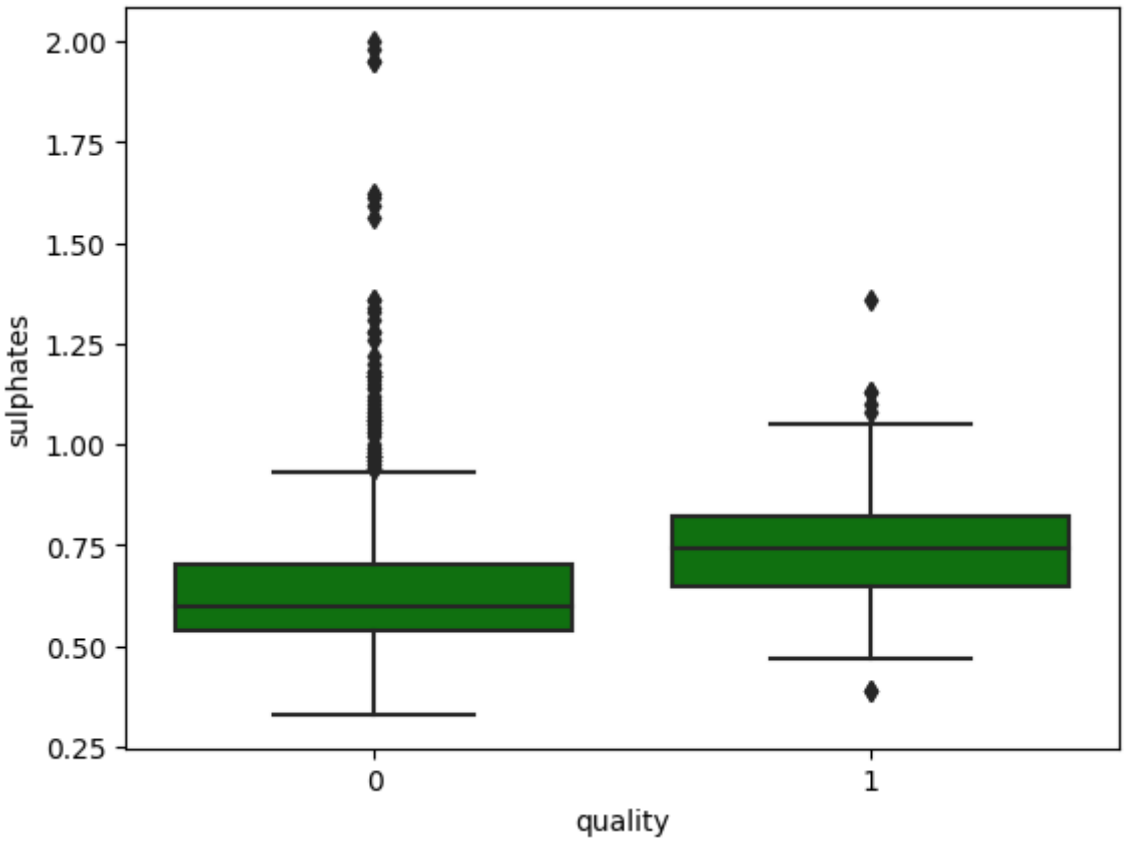
In [73]:
```
sns.displot(data = red_wine , x = 'sulphates')
```

Out[73]:
```
<seaborn.axisgrid.FacetGrid at 0x18692c10910>
```



In [74]:
```
sns.boxplot(data = red_wine , x = 'quality' , y = 'sulphates' , color='g')
```

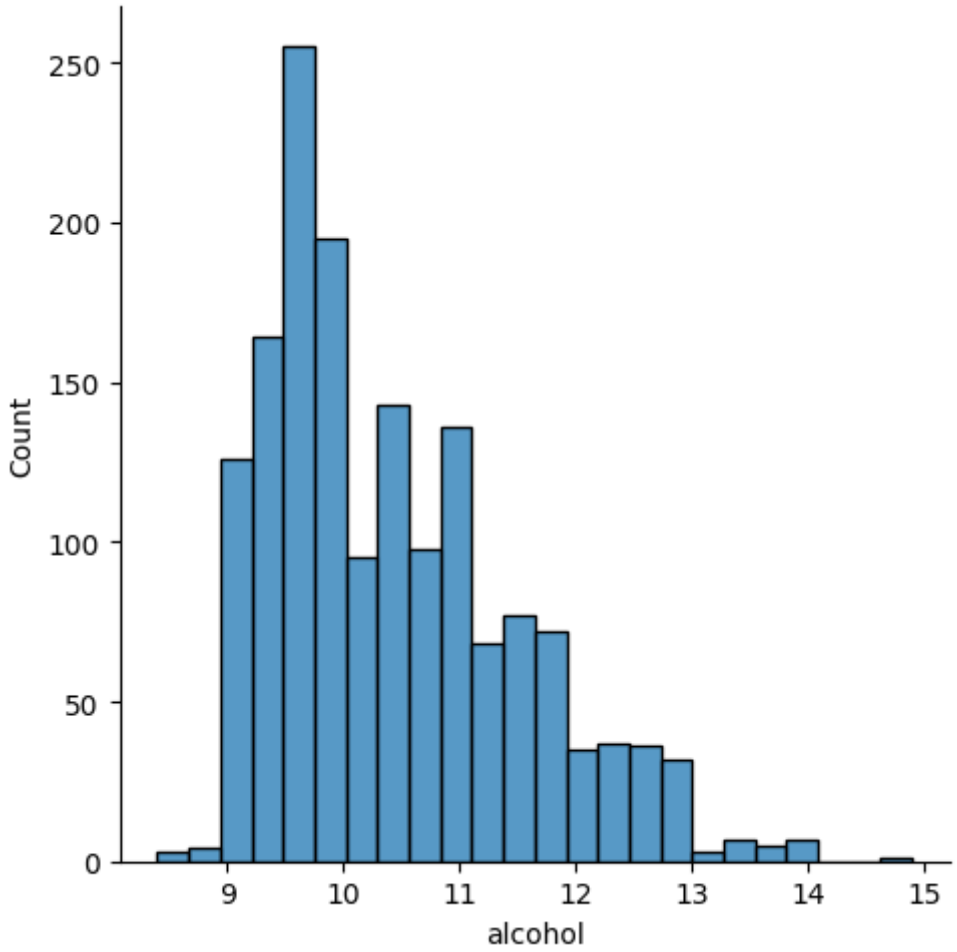
Out[74]:
```
<AxesSubplot: xlabel='quality', ylabel='sulphates'>
```



12) alcohol

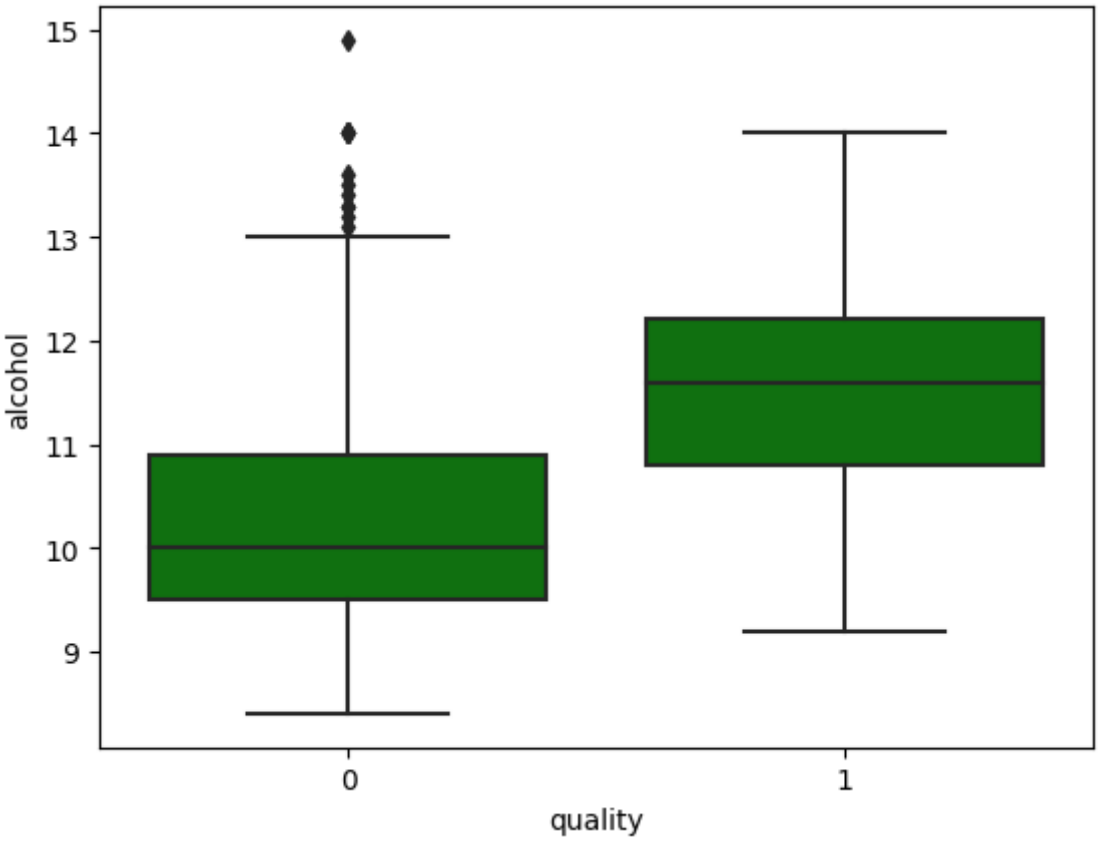
In [75]:
```
sns.displot(data = red_wine , x = 'alcohol')
```

Out[75]:
```
<seaborn.axisgrid.FacetGrid at 0x18692bbfe50>
```



In [76]:
```
sns.boxplot(data = red_wine , x = 'quality' , y = 'alcohol' , color='g')
```

Out[76]:
```
<AxesSubplot: xlabel='quality', ylabel='alcohol'>
```



Classification Models

In [77]:
```
red_wine.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   int32  
dtypes: float64(11), int32(1)
memory usage: 143.8 KB
```

In [78]:
```
X = red_wine.iloc[: , :11]
y = red_wine.iloc[: , -1]
```

In [79]:
```
# Splitting the dataset into the Training set and Test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_
```

In [80]:
```
print("Shape of X_train: ",X_train.shape)
print("Shape of X_test: ", X_test.shape)
```

```
print("Shape of y_train: ",y_train.shape)
print("Shape of y_test",y_test.shape)
```

```
Shape of X_train:  (1279, 11)
Shape of X_test:  (320, 11)
Shape of y_train:  (1279,)
Shape of y_test (320,)
```

In [81]:
```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_scaled = sc.fit_transform(X_train)
X_test_scaled = sc.transform(X_test)
```

1) Logistic Regression

In [82]:
```
# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
classifier_lr = LogisticRegression(C=1, fit_intercept=True, max_iter=1000, penalty
classifier_lr.fit(X_train_scaled, y_train.ravel())
```

Out[82]:
```
LogisticRegression(C=1, max_iter=1000, solver='liblinear')
```

In [84]:
```
# Predicting Cross Validation Score
from sklearn.model_selection import cross_val_score
from sklearn.metrics import confusion_matrix

cv_lr = cross_val_score(estimator = classifier_lr, X = X_train_scaled, y = y_train
print("CV: ", cv_lr.mean())

y_pred_lr_train = classifier_lr.predict(X_train_scaled)
accuracy_lr_train = accuracy_score(y_train, y_pred_lr_train)
print("Training set: ", accuracy_lr_train)

y_pred_lr_test = classifier_lr.predict(X_test_scaled)
accuracy_lr_test = accuracy_score(y_test, y_pred_lr_test)
print("Test set: ", accuracy_lr_test)
```

```
CV:  0.885857529527559
Training set:  0.8858483189992181
Test set:  0.865625
```

In [85]:
```
confusion_matrix(y_test, y_pred_lr_test)
```

Out[85]:
```
array([[264,   9],
       [ 34,  13]], dtype=int64)
```

In [86]:
```
from sklearn.metrics import classification_report

print(classification_report(y_test , y_pred_lr_test))
```

```
              precision    recall  f1-score   support

           0       0.89      0.97      0.92       273
           1       0.59      0.28      0.38        47

    accuracy                           0.87       320
   macro avg       0.74      0.62      0.65       320
weighted avg       0.84      0.87      0.84       320
```

In [87]:
```
tp_lr = confusion_matrix(y_test, y_pred_lr_test)[0,0]
fp_lr = confusion_matrix(y_test, y_pred_lr_test)[0,1]
```

```
tn_lr = confusion_matrix(y_test, y_pred_lr_test)[1,1]
fn_lr = confusion_matrix(y_test, y_pred_lr_test)[1,0]
```

2) K-Nearest Neighbors (K-NN)

In [88]:
```
# Fitting classifier to the Training set
from sklearn.neighbors import KNeighborsClassifier
classifier_knn = KNeighborsClassifier(leaf_size = 1, metric = 'minkowski', n_neighl
classifier_knn.fit(X_train_scaled, y_train.ravel())
```

Out[88]:
```
KNeighborsClassifier(leaf_size=1, n_neighbors=32, weights='distance')
```

In [92]:
```
y_train.ravel().shape
```

Out[92]:
```
(1279,)
```

In [93]:
```
# Predicting Cross Validation Score
cv_knn = cross_val_score(estimator = classifier_knn, X = X_train_scaled, y = y_tra
print("CV: ", cv_knn.mean())

y_pred_knn_train = classifier_knn.predict(X_train_scaled)
accuracy_knn_train = accuracy_score(y_train, y_pred_knn_train)
print("Training set: ", accuracy_knn_train)

y_pred_knn_test = classifier_knn.predict(X_test_scaled)
accuracy_knn_test = accuracy_score(y_test, y_pred_knn_test)
print("Test set: ", accuracy_knn_test)
```

```
CV:  0.9022699311023622
Training set:  1.0
Test set:  0.89375
```

In [94]:
```
confusion_matrix(y_test, y_pred_knn_test)
```

Out[94]:
```
array([[264,   9],
       [ 25,  22]], dtype=int64)
```

In [95]:
```
from sklearn.metrics import classification_report

print(classification_report(y_test , y_pred_knn_test))
```

```
              precision    recall  f1-score   support

           0       0.91      0.97      0.94       273
           1       0.71      0.47      0.56        47

    accuracy                           0.89       320
   macro avg       0.81      0.72      0.75       320
weighted avg       0.88      0.89      0.88       320
```

In [96]:
```
tp_knn = confusion_matrix(y_test, y_pred_knn_test)[0,0]
fp_knn = confusion_matrix(y_test, y_pred_knn_test)[0,1]
tn_knn = confusion_matrix(y_test, y_pred_knn_test)[1,1]
fn_knn = confusion_matrix(y_test, y_pred_knn_test)[1,0]
```

3) Support Vector Machine (SVM - Linear)

In [97]:
```
# Fitting classifier to the Training set
from sklearn.svm import SVC
```

```
classifier_svm_linear = SVC(kernel = 'linear')
classifier_svm_linear.fit(X_train_scaled, y_train.ravel())
```

Out[97]:
```
SVC(kernel='linear')
```

In [98]:
```
# Predicting Cross Validation Score
cv_svm_linear = cross_val_score(estimator = classifier_svm_linear, X = X_train_sca
print("CV: ", cv_svm_linear.mean())

y_pred_svm_linear_train = classifier_svm_linear.predict(X_train_scaled)
accuracy_svm_linear_train = accuracy_score(y_train, y_pred_svm_linear_train)
print("Training set: ", accuracy_svm_linear_train)

y_pred_svm_linear_test = classifier_svm_linear.predict(X_test_scaled)
accuracy_svm_linear_test = accuracy_score(y_test, y_pred_svm_linear_test)
print("Test set: ", accuracy_svm_linear_test)
```

```
CV:  0.8670829232283465
Training set:  0.8670836591086787
Test set:  0.853125
```

In [99]:
```
confusion_matrix(y_test, y_pred_svm_linear_test)
```

Out[99]:
```
array([[273,   0],
       [ 47,   0]], dtype=int64)
```

In [100…
```
from sklearn.metrics import classification_report

print(classification_report(y_test , y_pred_svm_linear_test))
```

```
              precision    recall  f1-score   support

           0       0.85      1.00      0.92       273
           1       0.00      0.00      0.00        47

    accuracy                           0.85       320
   macro avg       0.43      0.50      0.46       320
weighted avg       0.73      0.85      0.79       320
```

In [121…
```
tp_svm_linear = confusion_matrix(y_test, y_pred_svm_linear_test)[0,0]
fp_svm_linear = confusion_matrix(y_test, y_pred_svm_linear_test)[0,1]
tn_svm_linear = confusion_matrix(y_test, y_pred_svm_linear_test)[1,1]
fn_svm_linear = confusion_matrix(y_test, y_pred_svm_linear_test)[1,0]
```

Support Vector Machine (SVM - Kernel)

In [101…
```
# Fitting classifier to the Training set
from sklearn.svm import SVC
classifier_svm_kernel = SVC(kernel = 'rbf', C = 10, tol = 0.001, gamma = 'scale')
classifier_svm_kernel.fit(X_train_scaled, y_train.ravel())
```

Out[101]:
```
SVC(C=10)
```

In [102…
```
# Predicting Cross Validation Score
cv_svm_kernel = cross_val_score(estimator = classifier_svm_kernel, X = X_train_sca
print("CV: ", cv_svm_kernel.mean())

y_pred_svm_kernel_train = classifier_svm_kernel.predict(X_train_scaled)
accuracy_svm_kernel_train = accuracy_score(y_train, y_pred_svm_kernel_train)
print("Training set: ", accuracy_svm_kernel_train)
```

```
y_pred_svm_kernel_test = classifier_svm_kernel.predict(X_test_scaled)
accuracy_svm_kernel_test = accuracy_score(y_test, y_pred_svm_kernel_test)
print("Test set: ", accuracy_svm_kernel_test)
```

```
CV:  0.8999261811023622
Training set:  0.9421422986708365
Test set:  0.89375
```

In [103…
```
confusion_matrix(y_test, y_pred_svm_kernel_test)
```

Out[103]:
```
array([[261,  12],
       [ 22,  25]], dtype=int64)
```

In [105…
```
from sklearn.metrics import classification_report

print(classification_report(y_test , y_pred_svm_kernel_test))
```

```
              precision    recall  f1-score   support

           0       0.92      0.96      0.94       273
           1       0.68      0.53      0.60        47

    accuracy                           0.89       320
   macro avg       0.80      0.74      0.77       320
weighted avg       0.89      0.89      0.89       320
```

In [104…
```
tp_svm_kernel = confusion_matrix(y_test, y_pred_svm_kernel_test)[0,0]
fp_svm_kernel = confusion_matrix(y_test, y_pred_svm_kernel_test)[0,1]
tn_svm_kernel = confusion_matrix(y_test, y_pred_svm_kernel_test)[1,1]
fn_svm_kernel = confusion_matrix(y_test, y_pred_svm_kernel_test)[1,0]
```

4) Naive Bayes

In [106…
```
# Fitting classifier to the Training set
from sklearn.naive_bayes import GaussianNB
classifier_nb = GaussianNB()
classifier_nb.fit(X_train_scaled, y_train.ravel())
```

Out[106]:
```
GaussianNB()
```

In [107…
```
# Predicting Cross Validation Score
cv_nb = cross_val_score(estimator = classifier_nb, X = X_train_scaled, y = y_train
print("CV: ", cv_nb.mean())

y_pred_nb_train = classifier_nb.predict(X_train_scaled)
accuracy_nb_train = accuracy_score(y_train, y_pred_nb_train)
print("Training set: ", accuracy_nb_train)

y_pred_nb_test = classifier_nb.predict(X_test_scaled)
accuracy_nb_test = accuracy_score(y_test, y_pred_nb_test)
print("Test set: ", accuracy_nb_test)
```

```
CV:  0.8373462106299213
Training set:  0.8389366692728695
Test set:  0.846875
```

In [108…
```
confusion_matrix(y_test, y_pred_nb_test)
```

Out[108]:
```
array([[234,  39],
       [ 10,  37]], dtype=int64)
```

In [109…
```
from sklearn.metrics import classification_report

print(classification_report(y_test , y_pred_nb_test))
```

```
              precision    recall  f1-score   support

           0       0.96      0.86      0.91       273
           1       0.49      0.79      0.60        47

    accuracy                           0.85       320
   macro avg       0.72      0.82      0.75       320
weighted avg       0.89      0.85      0.86       320
```

In [120…
```
tp_nb = confusion_matrix(y_test, y_pred_nb_test)[0,0]
fp_nb = confusion_matrix(y_test, y_pred_nb_test)[0,1]
tn_nb = confusion_matrix(y_test, y_pred_nb_test)[1,1]
fn_nb = confusion_matrix(y_test, y_pred_nb_test)[1,0]
```

5) Decision Tree Classification

In [110…
```
# Fitting classifier to the Training set
from sklearn.tree import DecisionTreeClassifier
classifier_dt = DecisionTreeClassifier(criterion = 'gini', max_features=6, max_lea
classifier_dt.fit(X_train_scaled, y_train.ravel())
```

Out[110]:
```
DecisionTreeClassifier(max_features=6, max_leaf_nodes=400, random_state=33)
```

In [111…
```
# Predicting Cross Validation Score
from sklearn.model_selection import cross_val_score

cv_dt = cross_val_score(estimator = classifier_dt, X = X_train_scaled, y = y_train
print("CV: ", cv_dt.mean())

y_pred_dt_train = classifier_dt.predict(X_train_scaled)
accuracy_dt_train = accuracy_score(y_train, y_pred_dt_train)
print("Training set: ", accuracy_dt_train)

y_pred_dt_test = classifier_dt.predict(X_test_scaled)
accuracy_dt_test = accuracy_score(y_test, y_pred_dt_test)
print("Test set: ", accuracy_dt_test)
```

```
CV:  0.8960014763779528
Training set:  1.0
Test set:  0.878125
```

In [112…
```
confusion_matrix(y_test, y_pred_dt_test)
```

Out[112]:
```
array([[252,  21],
       [ 18,  29]], dtype=int64)
```

In [113…
```
from sklearn.metrics import classification_report

print(classification_report(y_test , y_pred_dt_test))
```

	precision	recall	f1-score	support
0	0.93	0.92	0.93	273
1	0.58	0.62	0.60	47
accuracy			0.88	320
macro avg	0.76	0.77	0.76	320
weighted avg	0.88	0.88	0.88	320

In [119…
```
tp_dt = confusion_matrix(y_test, y_pred_dt_test)[0,0]
fp_dt = confusion_matrix(y_test, y_pred_dt_test)[0,1]
tn_dt = confusion_matrix(y_test, y_pred_dt_test)[1,1]
fn_dt = confusion_matrix(y_test, y_pred_dt_test)[1,0]
```

6) Random Forest Classification

In [114…
```
# Fitting Random Forest Classification to the Training set
from sklearn.ensemble import RandomForestClassifier
classifier_rf = RandomForestClassifier(criterion = 'entropy', max_features = 4, n_
classifier_rf.fit(X_train_scaled, y_train.ravel())
```

Out[114]:
```
RandomForestClassifier(criterion='entropy', max_features=4, n_estimators=800,
                       random_state=33)
```

In [115…
```
# Predicting Cross Validation Score
cv_rf = cross_val_score(estimator = classifier_rf, X = X_train_scaled, y = y_train
print("CV: ", cv_rf.mean())

y_pred_rf_train = classifier_rf.predict(X_train_scaled)
accuracy_rf_train = accuracy_score(y_train, y_pred_rf_train)
print("Training set: ", accuracy_rf_train)

y_pred_rf_test = classifier_rf.predict(X_test_scaled)
accuracy_rf_test = accuracy_score(y_test, y_pred_rf_test)
print("Test set: ", accuracy_rf_test)
```

```
CV:  0.9140194389763779
Training set:  1.0
Test set:  0.9125
```

In [116…
```
confusion_matrix(y_test, y_pred_rf_test)
```

Out[116]:
```
array([[267,   6],
       [ 22,  25]], dtype=int64)
```

In [117…
```
from sklearn.metrics import classification_report

print(classification_report(y_test , y_pred_rf_test))
```

	precision	recall	f1-score	support
0	0.92	0.98	0.95	273
1	0.81	0.53	0.64	47
accuracy			0.91	320
macro avg	0.87	0.75	0.80	320
weighted avg	0.91	0.91	0.90	320

In [118…
```
tp_rf = confusion_matrix(y_test, y_pred_rf_test)[0,0]
fp_rf = confusion_matrix(y_test, y_pred_rf_test)[0,1]
```

```
tn_rf = confusion_matrix(y_test, y_pred_rf_test)[1,1]
fn_rf = confusion_matrix(y_test, y_pred_rf_test)[1,0]
```

Models summary

In [122…

```
models = [('Logistic Regression', tp_lr, fp_lr, tn_lr, fn_lr, accuracy_lr_train, a
          ('K-Nearest Neighbors (KNN)', tp_knn, fp_knn, tn_knn, fn_knn, accuracy_k
          ('SVM (Linear)', tp_svm_linear, fp_svm_linear, tn_svm_linear, fn_svm_lin
          ('SVM (Kernel)', tp_svm_kernel, fp_svm_kernel, tn_svm_kernel, fn_svm_ker
          ('Naive Bayes', tp_nb, fp_nb, tn_nb, fn_nb, accuracy_nb_train, accuracy_
          ('Decision Tree Classification', tp_dt, fp_dt, tn_dt, fn_dt, accuracy_dt_
          ('Random Forest Tree Classification', tp_rf, fp_rf, tn_rf, fn_rf, accura
         ]
```

In [123…

```
predict = pd.DataFrame(data = models, columns=['Model', 'True Positive', 'False Pos
                                               'False Negative', 'Accuracy(training
                                               'Cross-Validation'])
predict
```

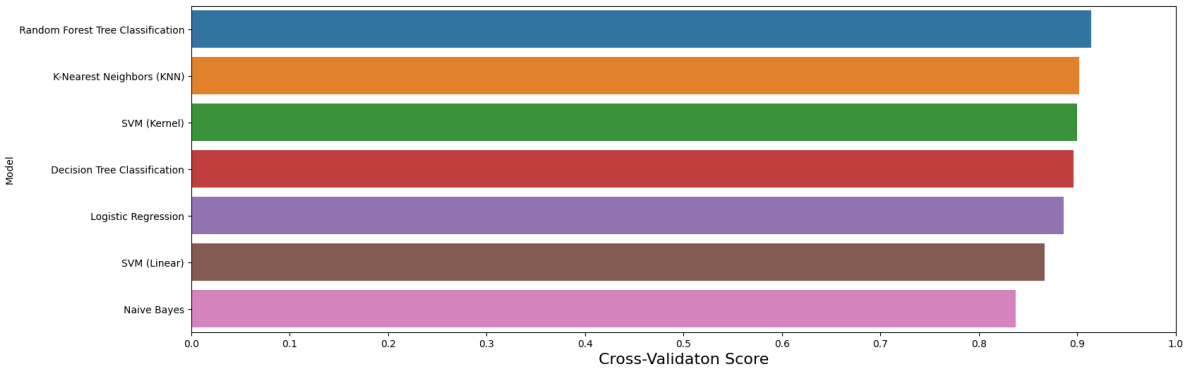
Out[123]:

	Model	True Positive	False Positive	True Negative	False Negative	Accuracy(training)	Accuracy(test)	Cros Validati
0	Logistic Regression	264	9	13	34	0.885848	0.865625	0.8858
1	K-Nearest Neighbors (KNN)	264	9	22	25	1.000000	0.893750	0.9022
2	SVM (Linear)	273	0	0	47	0.867084	0.853125	0.8670
3	SVM (Kernel)	261	12	25	22	0.942142	0.893750	0.8999
4	Naive Bayes	234	39	37	10	0.838937	0.846875	0.8373
5	Decision Tree Classification	252	21	29	18	1.000000	0.878125	0.8960
6	Random Forest Tree Classification	267	6	25	22	1.000000	0.912500	0.9140

Visualizing Models Performance ¶

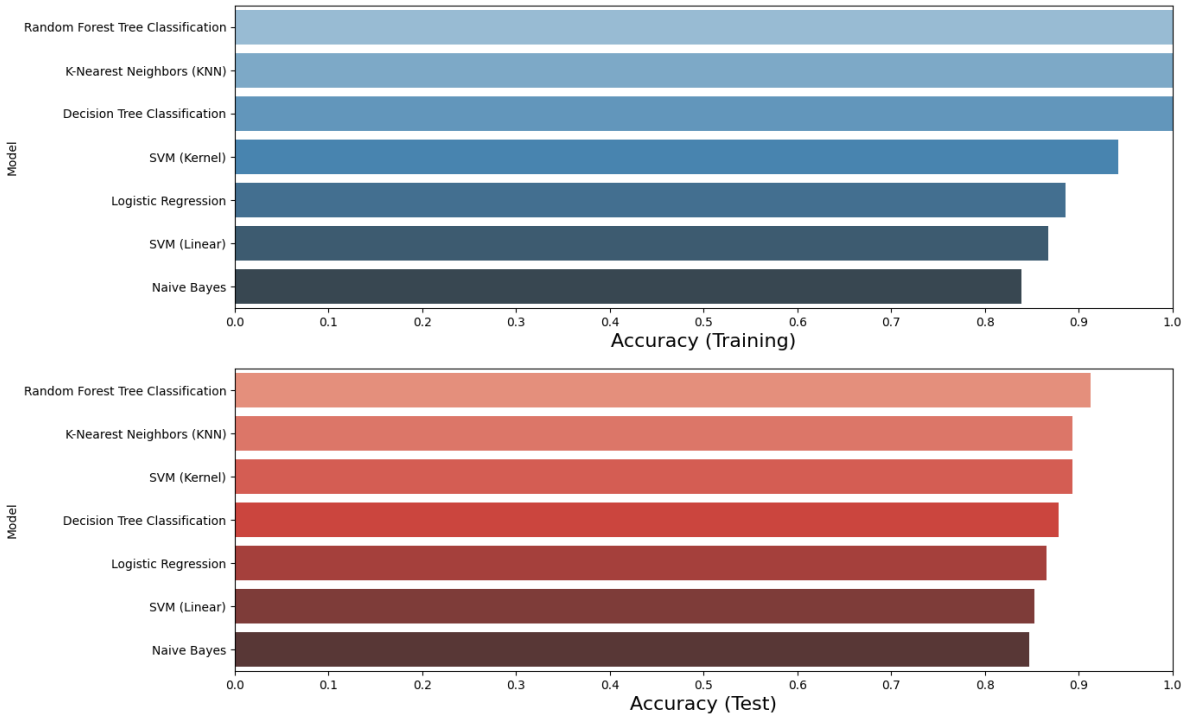
In [124…

```
f, axe = plt.subplots(1,1, figsize=(18,6))

predict.sort_values(by=['Cross-Validation'], ascending=False, inplace=True)

sns.barplot(x='Cross-Validation', y='Model', data = predict, ax = axe)
#axes[0].set(xlabel='Region', ylabel='Charges')
axe.set_xlabel('Cross-Validaton Score', size=16)
axe.set_ylabel('Model')
axe.set_xlim(0,1.0)
axe.set_xticks(np.arange(0, 1.1, 0.1))
plt.show()
```



In [125…

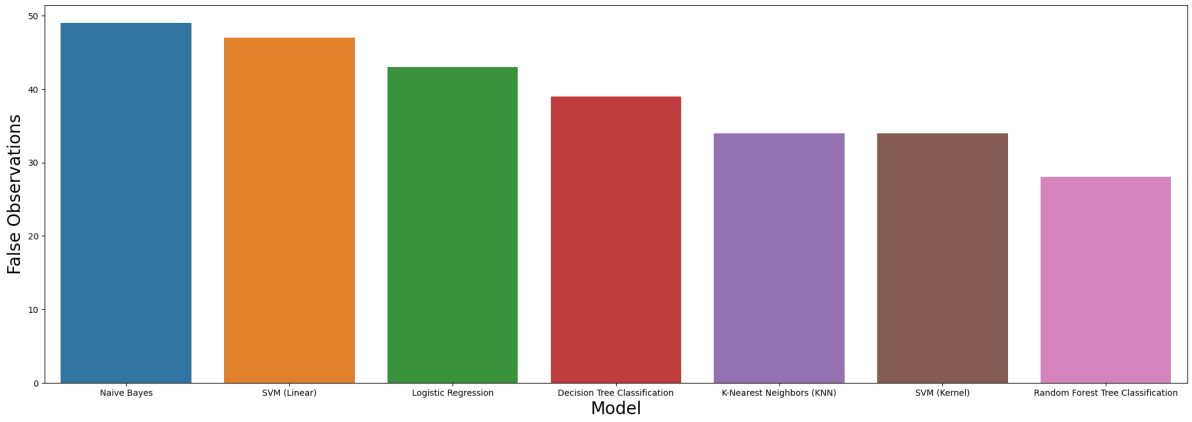
```
f, axes = plt.subplots(2,1, figsize=(14,10))

predict.sort_values(by=['Accuracy(training)'], ascending=False, inplace=True)

sns.barplot(x='Accuracy(training)', y='Model', data = predict, palette='Blues_d', a
#axes[0].set(xlabel='Region', ylabel='Charges')
axes[0].set_xlabel('Accuracy (Training)', size=16)
axes[0].set_ylabel('Model')
axes[0].set_xlim(0,1.0)
axes[0].set_xticks(np.arange(0, 1.1, 0.1))

predict.sort_values(by=['Accuracy(test)'], ascending=False, inplace=True)

sns.barplot(x='Accuracy(test)', y='Model', data = predict, palette='Reds_d', ax = a
#axes[0].set(xlabel='Region', ylabel='Charges')
axes[1].set_xlabel('Accuracy (Test)', size=16)
axes[1].set_ylabel('Model')
axes[1].set_xlim(0,1.0)
axes[1].set_xticks(np.arange(0, 1.1, 0.1))

plt.show()
```



In [126…

```
predict.sort_values(by=(['Accuracy(test)']), ascending=True, inplace=True)

f, axe = plt.subplots(1,1, figsize=(24,8))
sns.barplot(x = predict['Model'], y=predict['False Positive'] + predict['False Nega
axe.set_xlabel('Model', size=20)
axe.set_ylabel('False Observations', size=20)
```

```
plt.show()
```


False Observations
0
10
20
30
40
50
Naive Bayes
SVM (Linear)
Logistic Regression
Decision Tree Classification
K-Nearest Neighbors (KNN)
SVM (Kernel)
Random Forest Tree Classification
Model

In [ ]: