In [1]:

```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```python
data =pd.read_csv("F:/winequality-red.csv")
data
```

Out[2]:

| | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | pH | sulphates | alcohol | quality |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 7.4 | 0.700 | 0.00 | 1.9 | 0.076 | 11.0 | 34.0 | 0.99780 | 3.51 | 0.56 | 9.4 | 5 |
| **1** | 7.8 | 0.880 | 0.00 | 2.6 | 0.098 | 25.0 | 67.0 | 0.99680 | 3.20 | 0.68 | 9.8 | 5 |
| **2** | 7.8 | 0.760 | 0.04 | 2.3 | 0.092 | 15.0 | 54.0 | 0.99700 | 3.26 | 0.65 | 9.8 | 5 |
| **3** | 11.2 | 0.280 | 0.56 | 1.9 | 0.075 | 17.0 | 60.0 | 0.99800 | 3.16 | 0.58 | 9.8 | 6 |
| **4** | 7.4 | 0.700 | 0.00 | 1.9 | 0.076 | 11.0 | 34.0 | 0.99780 | 3.51 | 0.56 | 9.4 | 5 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **1594** | 6.2 | 0.600 | 0.08 | 2.0 | 0.090 | 32.0 | 44.0 | 0.99490 | 3.45 | 0.58 | 10.5 | 5 |
| **1595** | 5.9 | 0.550 | 0.10 | 2.2 | 0.062 | 39.0 | 51.0 | 0.99512 | 3.52 | 0.76 | 11.2 | 6 |
| **1596** | 6.3 | 0.510 | 0.13 | 2.3 | 0.076 | 29.0 | 40.0 | 0.99574 | 3.42 | 0.75 | 11.0 | 6 |
| **1597** | 5.9 | 0.645 | 0.12 | 2.0 | 0.075 | 32.0 | 44.0 | 0.99547 | 3.57 | 0.71 | 10.2 | 5 |
| **1598** | 6.0 | 0.310 | 0.47 | 3.6 | 0.067 | 18.0 | 42.0 | 0.99549 | 3.39 | 0.66 | 11.0 | 6 |

1599 rows × 12 columns

In [3]:

```python
data.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 [4]:

```python
data.describe()
```

Out[4]:

| | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | pH | s |
|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 1599.000000 | 1599.000000 | 1599.000000 | 1599.000000 | 1599.000000 | 1599.000000 | 1599.000000 | 1599.000000 | 1599.000000 | 159 |
| **mean** | 8.319637 | 0.527821 | 0.270976 | 2.538806 | 0.087467 | 15.874922 | 46.467792 | 0.996747 | 3.311113 | |
| **std** | 1.741096 | 0.179060 | 0.194801 | 1.409928 | 0.047065 | 10.460157 | 32.895324 | 0.001887 | 0.154386 | |
| **min** | 4.600000 | 0.120000 | 0.000000 | 0.900000 | 0.012000 | 1.000000 | 6.000000 | 0.990070 | 2.740000 | |
| **25%** | 7.100000 | 0.390000 | 0.090000 | 1.900000 | 0.070000 | 7.000000 | 22.000000 | 0.995600 | 3.210000 | |
| **50%** | 7.900000 | 0.520000 | 0.260000 | 2.200000 | 0.079000 | 14.000000 | 38.000000 | 0.996750 | 3.310000 | |
| **75%** | 9.200000 | 0.640000 | 0.420000 | 2.600000 | 0.090000 | 21.000000 | 62.000000 | 0.997835 | 3.400000 | |
| **max** | 15.900000 | 1.580000 | 1.000000 | 15.500000 | 0.611000 | 72.000000 | 289.000000 | 1.003690 | 4.010000 | |

In [5]:

```python
data.isna().sum()
```

Out[5]:

```
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 [6]:

```python
data.rename(columns={
    "fixed acidity": "fixed_acidity",
                   "volatile acidity": "volatile_acidity",
                   "citric acid": "citric_acid",
                   "residual sugar": "residual_sugar",
                   "chlorides": "chlorides",
                   "free sulfur dioxide": "free_sulfur_dioxide",
                   "total sulfur dioxide": "total_sulfur_dioxide"
},inplace=True)
```

In [7]:

```python
data.columns
```

Out[7]:

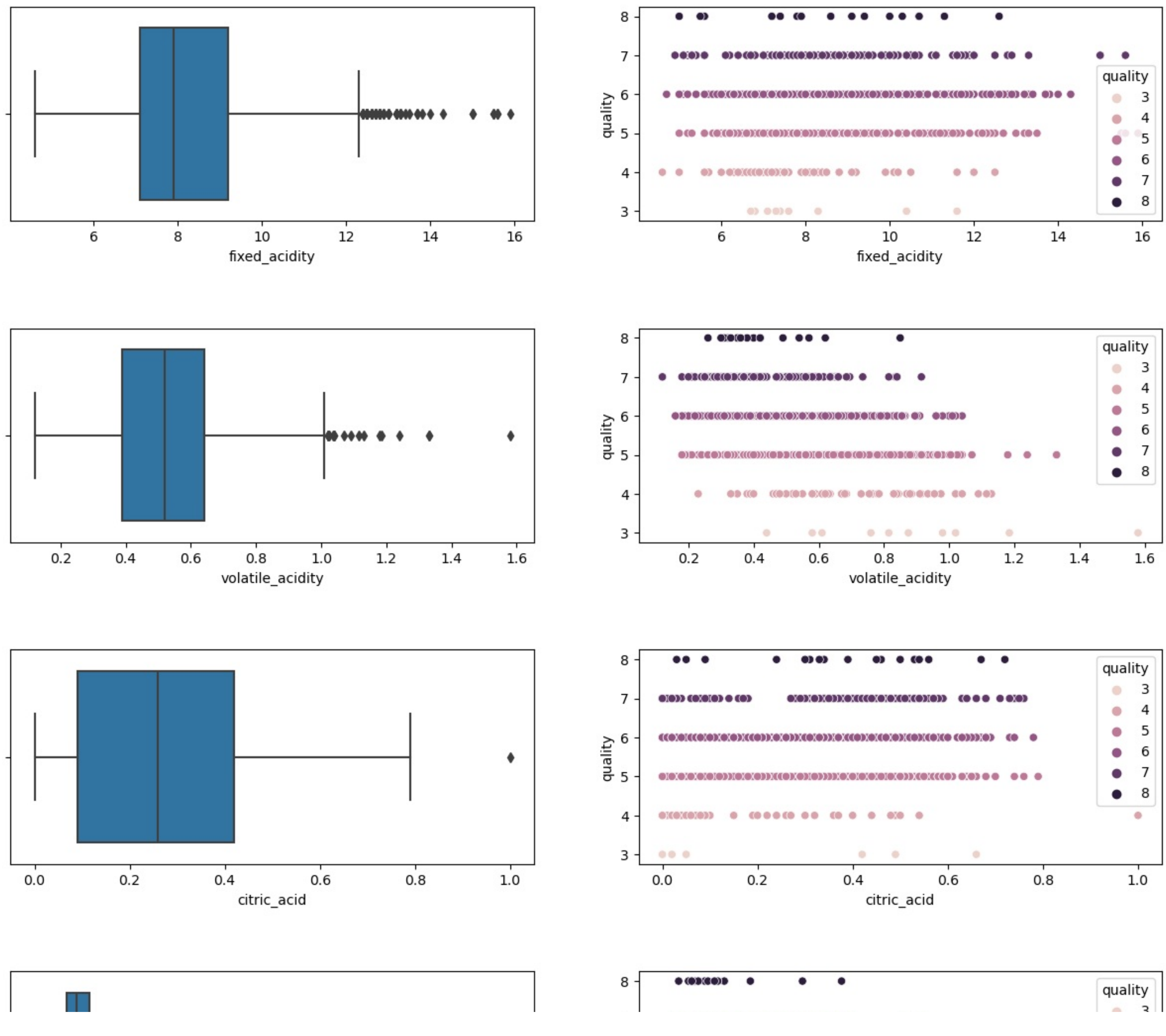
```
Index(['fixed_acidity', 'volatile_acidity', 'citric_acid', 'residual_sugar',
       'chlorides', 'free_sulfur_dioxide', 'total_sulfur_dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

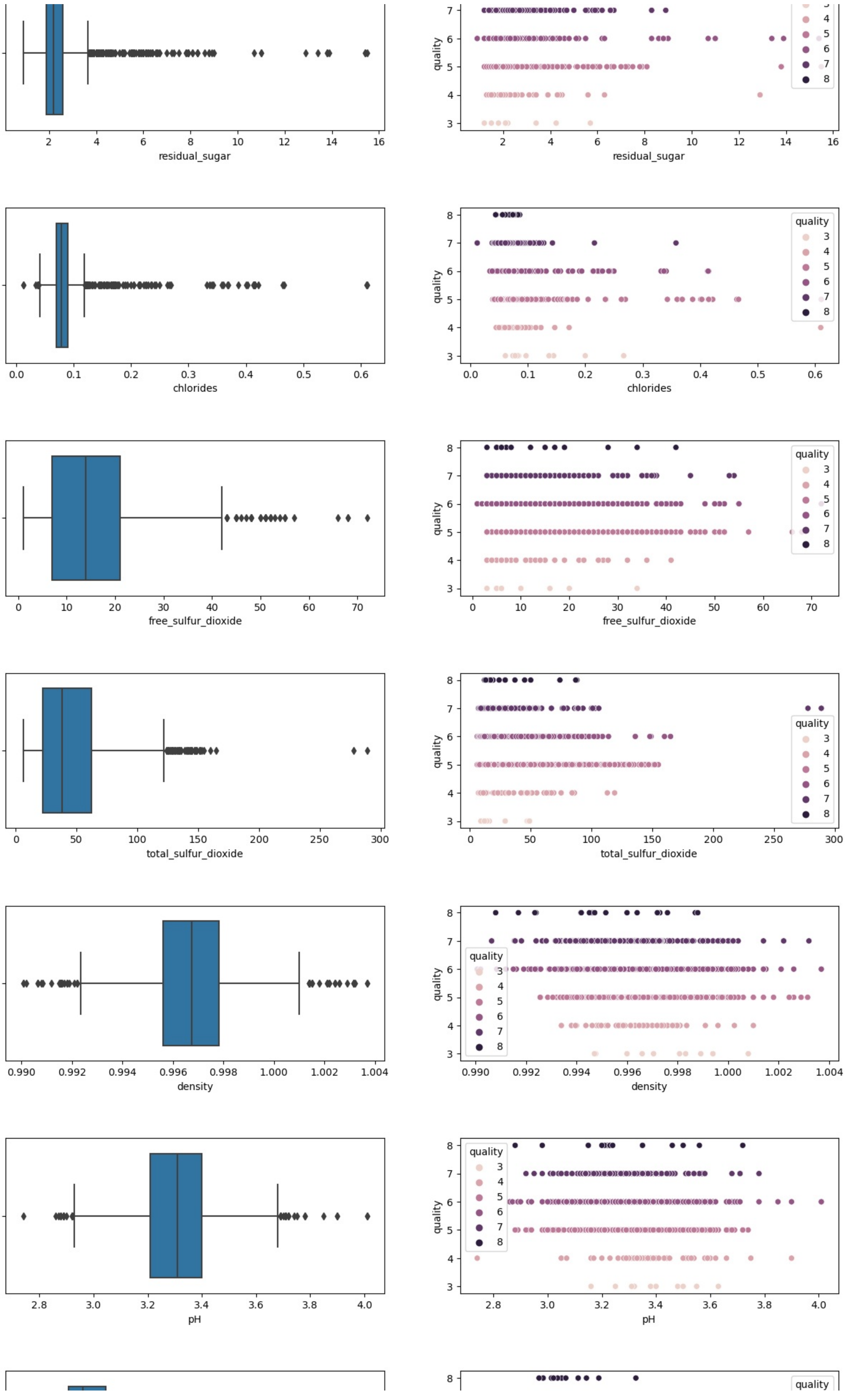
In [8]:

```python
columns=list(data.columns)
```

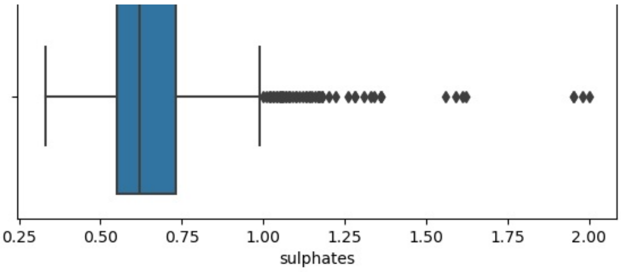
In [9]:

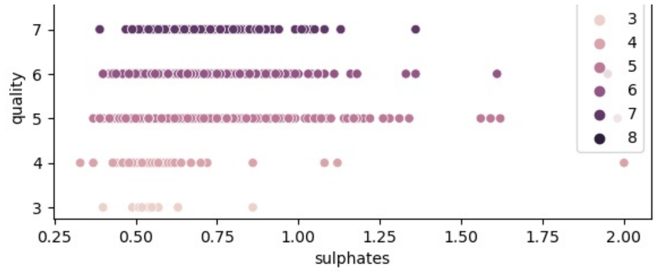
```python
fig,ax =plt.subplots(11,2,figsize=(15,45))
plt.subplots_adjust(hspace=0.5)
for i in range(11):
    sns.boxplot(x=columns[i],data=data,ax=ax[i,0])
    sns.scatterplot(x=columns[i],y="quality",data=data,hue="quality",ax=ax[i,1])
```

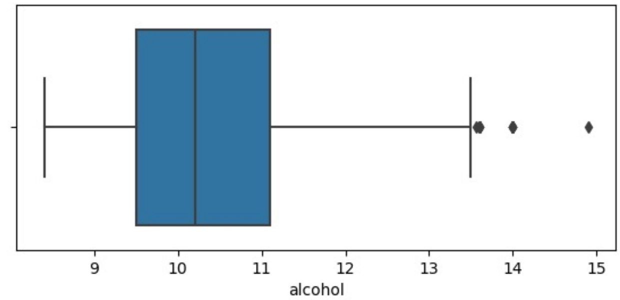


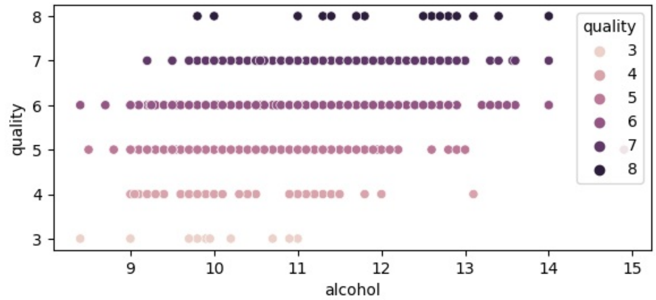


residual_sugar
quality
chlorides
free_sulfur_dioxide
total_sulfur_dioxide
density
pH

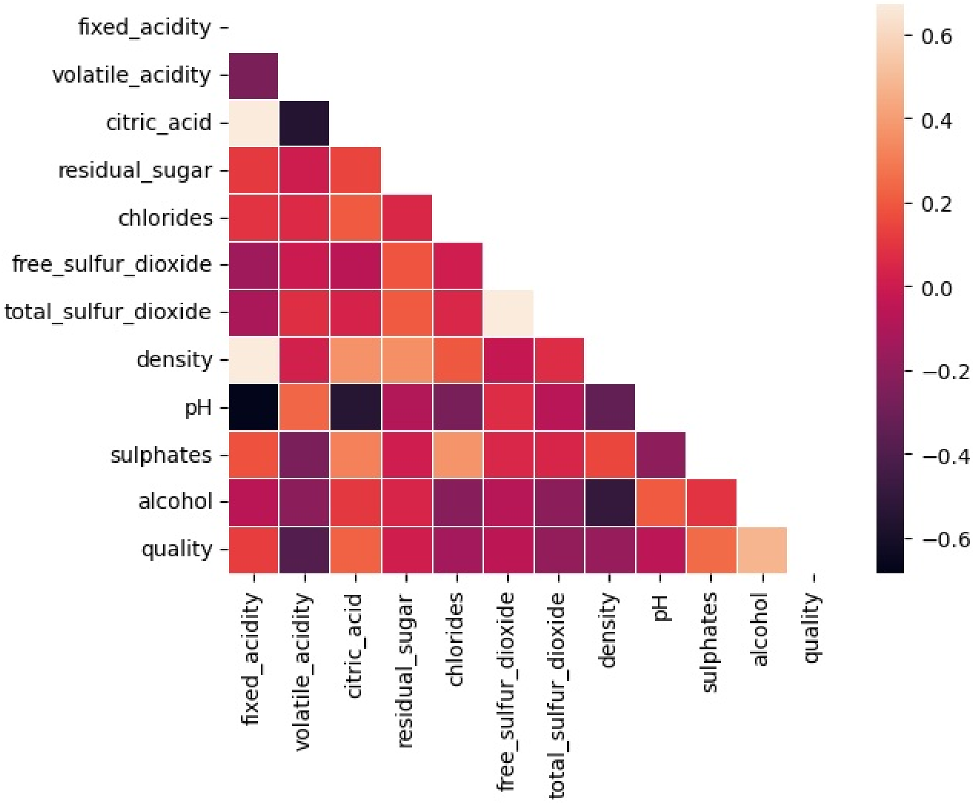


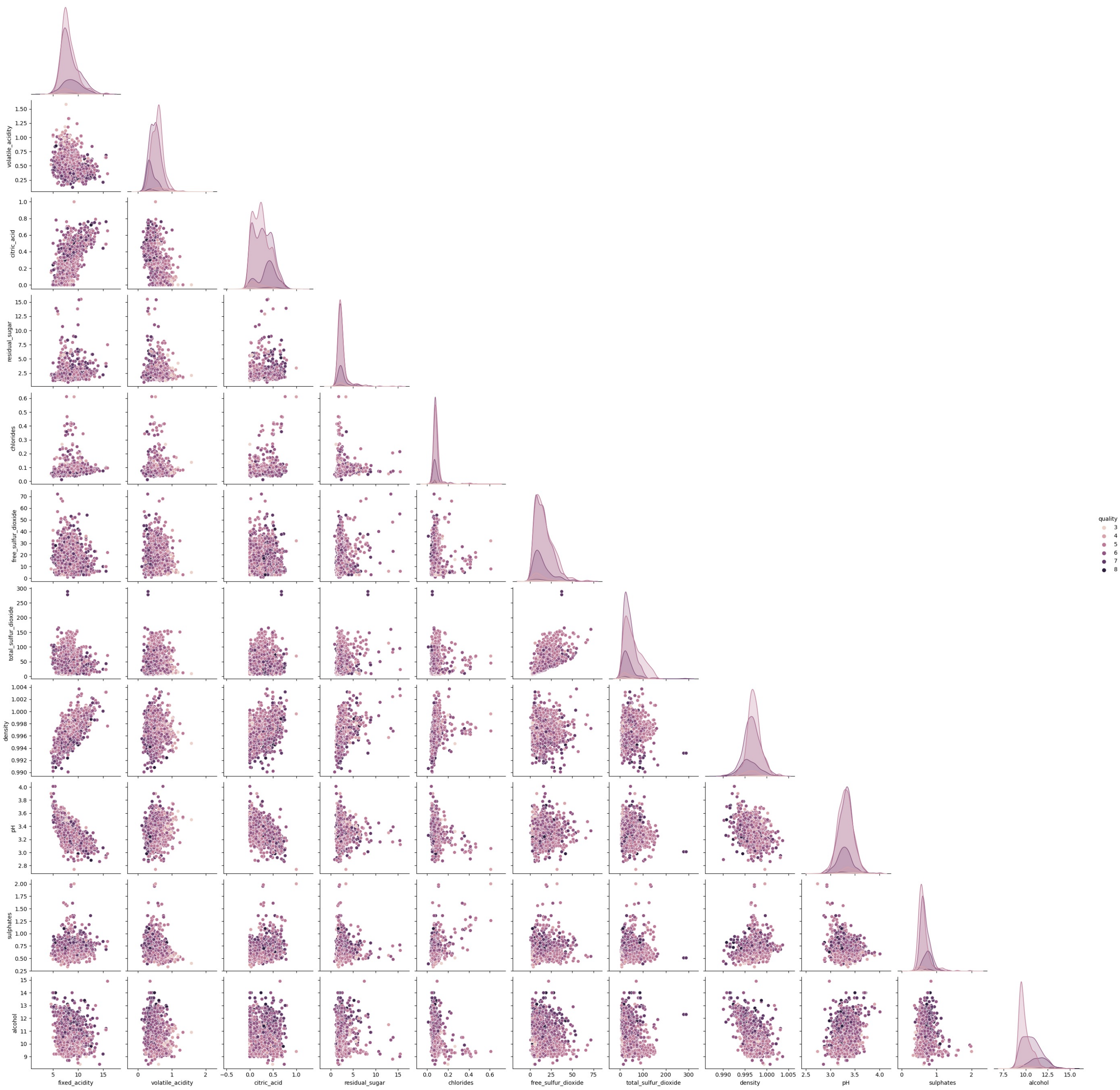




In [10]:
```
corr=data.corr()
sns.heatmap(corr,annot=True,linewidth=0.5,mask=np.triu(corr))
plt.show()
```

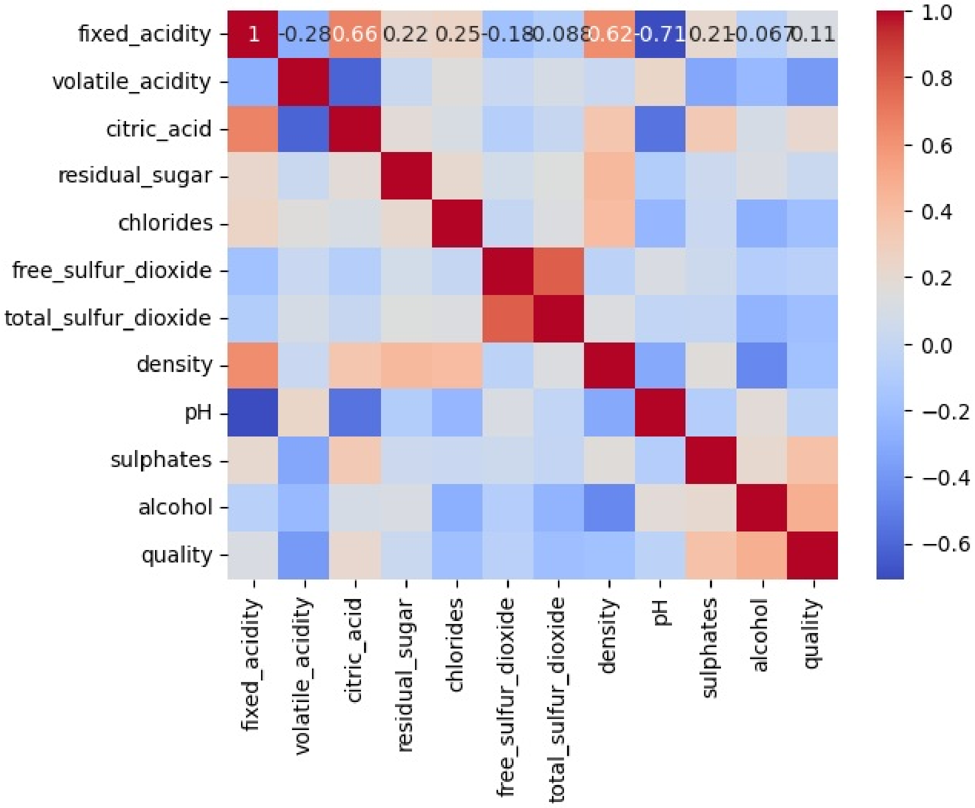



In [11]:
```
sns.pairplot(data,hue="quality",corner=True)
```

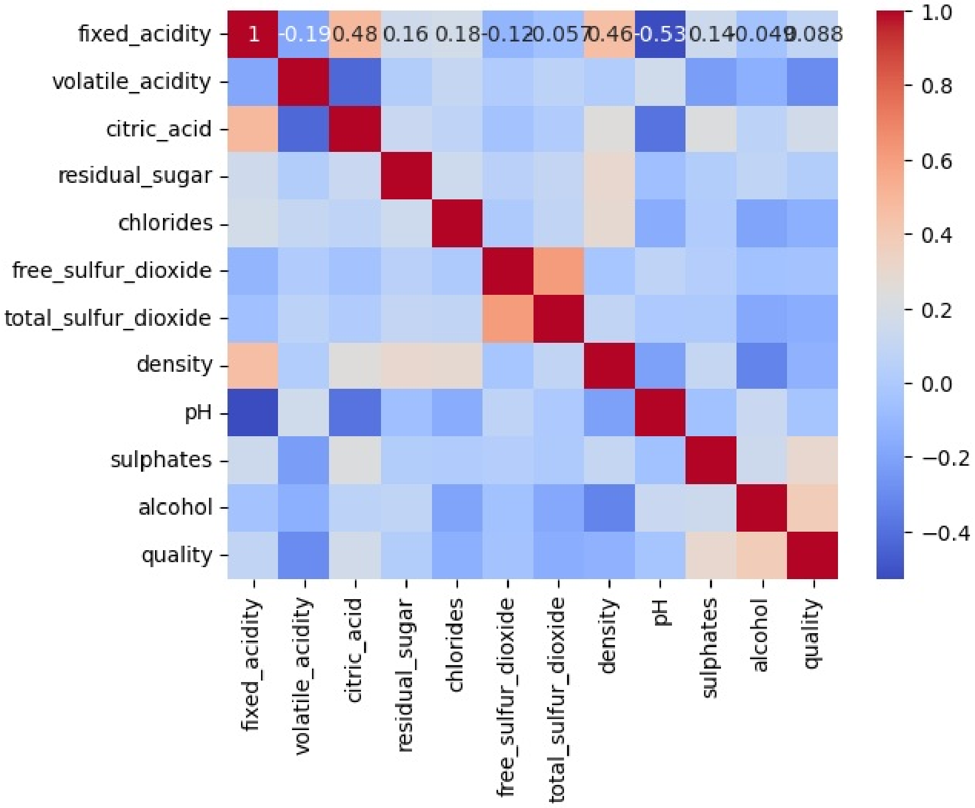
Out[11]:
```
<seaborn.axisgrid.PairGrid at 0x1e0449879d0>
```



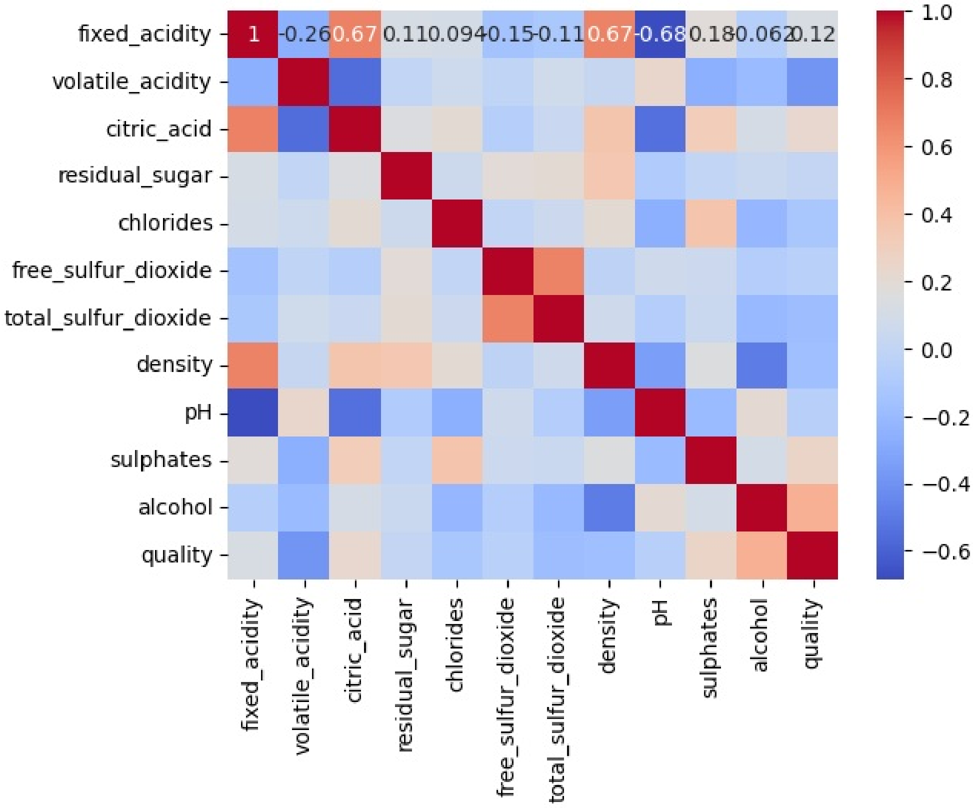

In [12]:
```
correlation_matrix = data.corr(method='spearman')
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.show()
```



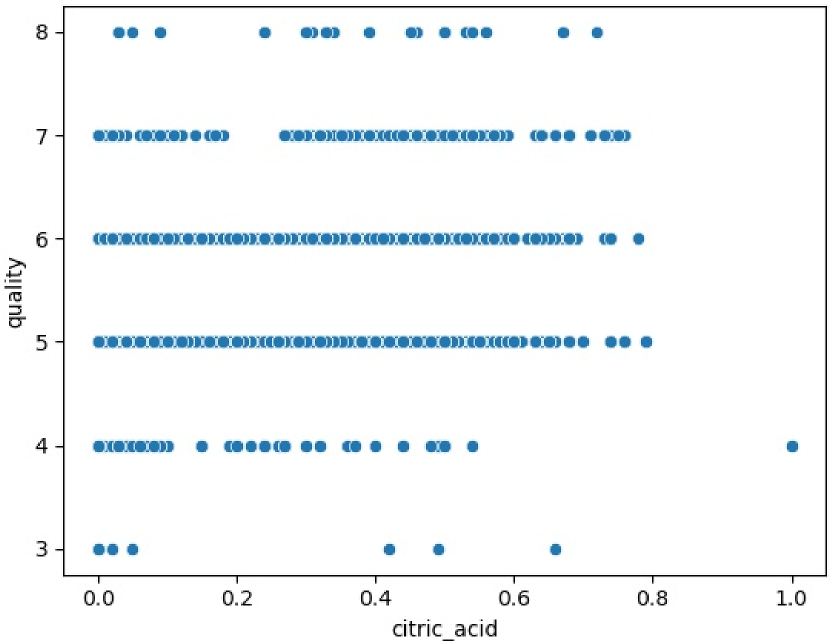

In [13]:
```
correlation_matrix = data.corr(method='kendall')
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.show()
```




In [14]:
```
correlation_matrix = data.corr(method='pearson')
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.show()
```




In [15]:
```
sns.scatterplot(x='citric_acid', y='quality', data=data)
plt.show()
```

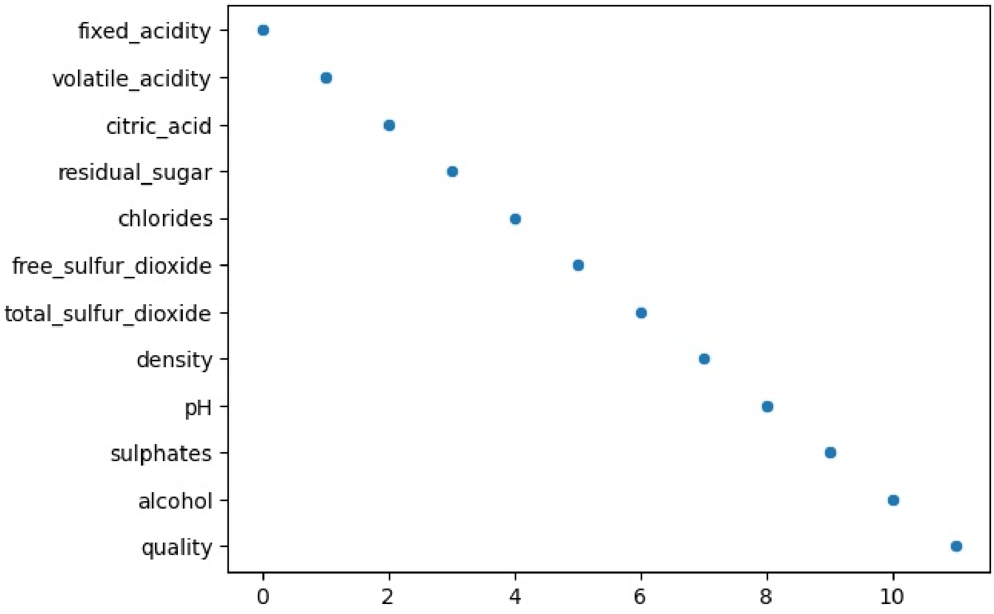



In [16]:
```
# Pearson Correlation
pearson_corr = data['fixed_acidity'].corr(data['citric_acid'])
print(f'Pearson Correlation: {pearson_corr}')

# Spearman Correlation
spearman_corr = data['fixed_acidity'].corr(data['citric_acid'], method='spearman')
print(f'Spearman Correlation: {spearman_corr}')

# Kendall Correlation
kendall_corr = data['fixed_acidity'].corr(data['citric_acid'], method='kendall')
print(f'Kendall Correlation: {kendall_corr}')
```

```
Pearson Correlation: 0.6717034347641059
Spearman Correlation: 0.6617084166678848
Kendall Correlation: 0.484271229113174
```

In [17]:
```
sns.scatterplot(data=columns)
```

Out[17]: <Axes: >




In [18]:
```
data.quality.unique()
```

Out[18]:
```
array([5, 6, 7, 4, 8, 3], dtype=int64)
```

In [19]:
```
data=data.replace({"quality":{8 : 'Good',
                              7 : 'Good',
                              6 : 'Middle',
                              5 : 'Middle',
                              4 : 'Bad',
                              3 : 'Bad',}})
```

In [20]:
```
data
```

Out[20]:

| | fixed_acidity | volatile_acidity | citric_acid | residual_sugar | chlorides | free_sulfur_dioxide | total_sulfur_dioxide | density | pH | s |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 7.4 | 0.700 | 0.00 | 1.9 | 0.076 | 11.0 | 34.0 | 0.99780 | 3.51 | |
| **1** | 7.8 | 0.880 | 0.00 | 2.6 | 0.098 | 25.0 | 67.0 | 0.99680 | 3.20 | |
| **2** | 7.8 | 0.760 | 0.04 | 2.3 | 0.092 | 15.0 | 54.0 | 0.99700 | 3.26 | |
| **3** | 11.2 | 0.280 | 0.56 | 1.9 | 0.075 | 17.0 | 60.0 | 0.99800 | 3.16 | |
| **4** | 7.4 | 0.700 | 0.00 | 1.9 | 0.076 | 11.0 | 34.0 | 0.99780 | 3.51 | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **1594** | 6.2 | 0.600 | 0.08 | 2.0 | 0.090 | 32.0 | 44.0 | 0.99490 | 3.45 | |
| **1595** | 5.9 | 0.550 | 0.10 | 2.2 | 0.062 | 39.0 | 51.0 | 0.99512 | 3.52 | |
| **1596** | 6.3 | 0.510 | 0.13 | 2.3 | 0.076 | 29.0 | 40.0 | 0.99574 | 3.42 | |
| **1597** | 5.9 | 0.645 | 0.12 | 2.0 | 0.075 | 32.0 | 44.0 | 0.99547 | 3.57 | |
| **1598** | 6.0 | 0.310 | 0.47 | 3.6 | 0.067 | 18.0 | 42.0 | 0.99549 | 3.39 | |

1599 rows × 12 columns

In [21]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [22]:
```
X_temp=data.drop(columns="quality")
y=data.quality
```

In [23]:
```
scaler=MinMaxScaler(feature_range=(0,1)).fit_transform(X_temp)
X=pd.DataFrame(scaler,columns=X_temp.columns)
```

In [24]:
```
X.describe()
```

Out[24]:

| | fixed_acidity | volatile_acidity | citric_acid | residual_sugar | chlorides | free_sulfur_dioxide | total_sulfur_dioxide | densit |
|---|---|---|---|---|---|---|---|---|
| count | 1599.000000 | 1599.000000 | 1599.000000 | 1599.000000 | 1599.000000 | 1599.000000 | 1599.000000 | 1599.00000 |
| mean | 0.329171 | 0.279329 | 0.270976 | 0.112247 | 0.125988 | 0.209506 | 0.142996 | 0.49021 |
| std | 0.154079 | 0.122644 | 0.194801 | 0.096570 | 0.078573 | 0.147326 | 0.116238 | 0.13857 |
| min | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.00000 |
| 25% | 0.221239 | 0.184932 | 0.090000 | 0.068493 | 0.096828 | 0.084507 | 0.056537 | 0.40602 |
| 50% | 0.292035 | 0.273973 | 0.260000 | 0.089041 | 0.111853 | 0.183099 | 0.113074 | 0.49045 |
| 75% | 0.407080 | 0.356164 | 0.420000 | 0.116438 | 0.130217 | 0.281690 | 0.197880 | 0.57011 |
| max | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.00000 |

In [25]:
```
from sklearn.model_selection import train_test_split,GridSearchCV,cross_val_score,KFold
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
```

In [ ]:

In [26]:
```
def confusion_plot(y_test,y_prediction):
    cm=metrics.confusion_matrics(y_test,y_prediction)
    ax=plt.subplot()
    ax = sns.heatmap(cm, annot=True, fmt='', cmap="Purples")
    ax.set_xlabel('Prediced labels', fontsize=18)
    ax.set_ylabel('True labels', fontsize=18)
    ax.set_title('Confusion Matrix', fontsize=25)
    ax.xaxis.set_ticklabels(['Bad', 'Good', 'Middle'])
    ax.yaxis.set_ticklabels(['Bad', 'Good', 'Middle'])
    plt.show()
```

In [27]:
```
def clfr_plot(y_test,y_pred):
    cr=pd.DataFrame(metrics.Classification_report(y_test,y_pred_rf,digits=3,output_dict=True)).T
    cr.drop(columns="support",inplace=True)
    sns.heatmap(cr,annot=True,cmap="Purples",linecolor="white",linewidths="0.5").xaxis.tick_top()
```

In [28]:
```
def clf_plot(y_pred):
    cm=metrics.confussion_matrics(y_test,y_pred)
    cr=pd.DataFrame(metrics.classification_report(y_test,y_pred_rf,digits=3,output_dict=True)).T
    cr.drop(columns="suport",inplace=True)
    fig,ax=plt.subplot(1,2,figsize=(15,5))
    ax[0] = sns.heatmap(cm, annot=True, fmt='', cmap="Purples", ax=ax[0])
    ax[0].set_xlabel('Prediced labels', fontsize=18)
    ax[0].set_ylabel('True labels', fontsize=18)
    ax[0].set_title('Confusion Matrix', fontsize=25)
    ax[0].xaxis.set_ticklabels(['Bad', 'Good', 'Middle'])
    ax[0].yaxis.set_ticklabels(['Bad', 'Good', 'Middle'])

    # Right AX : Classification Report
    ax[1] = sns.heatmap(cr, cmap='Purples', annot=True, linecolor='white', linewidths=0.5, ax=ax[1])
    ax[1].xaxis.tick_top()
    ax[1].set_title('Classification Report', fontsize=25)
    plt.show()
```

In [29]:
```
data.quality.value_counts()
```

Out[29]:
```
quality
Middle    1319
Good       217
Bad         63
Name: count, dtype: int64
```

In [30]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.25,random_state=0)
```

In [31]:
```
from sklearn.ensemble import RandomForestClassifier
parameters = {
    'n_estimators': [50, 150, 500],
    'criterion': ['gini', 'entropy', 'log_loss'],
    'max_features': ['sqrt', 'log2']
}

# Initialize the classifier
```

```python
rf = RandomForestClassifier(n_jobs=-1)

# Perform grid search
rf_cv = GridSearchCV(estimator=rf, cv=20, param_grid=parameters)
rf_cv.fit(X_train, y_train)

# Print the results
print('Tuned hyper parameters:', rf_cv.best_params_)
print('Accuracy:', rf_cv.best_score_)
```

```
Tuned hyper parameters: {'criterion': 'gini', 'max_features': 'sqrt', 'n_estimators': 150}
Accuracy: 0.8657203389830508
```

In [32]:

```python
X_temp=data.drop(columns="quality")
y=data.quality
```

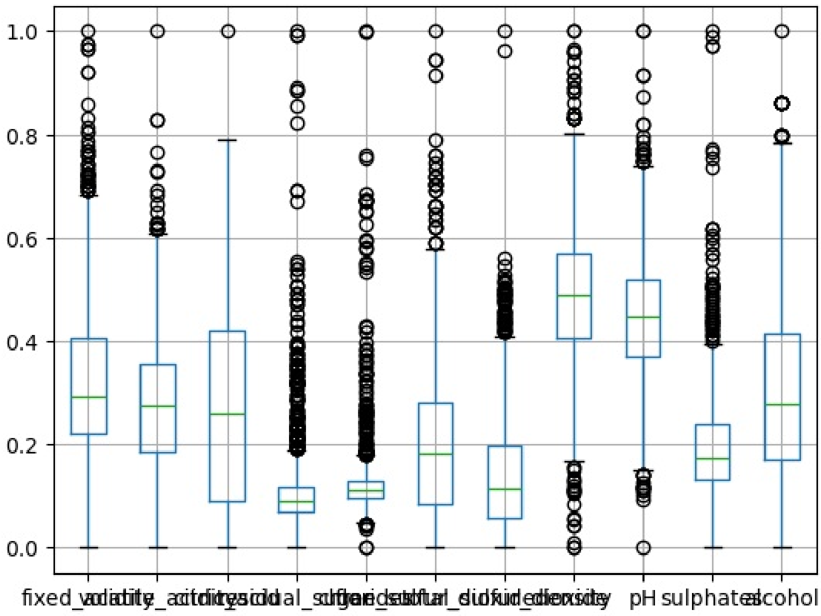
In [33]:

```python
scaler=MinMaxScaler(feature_range=(0,1)).fit_transform(X_temp)
X=pd.DataFrame(scaler,columns=X_temp.columns)
```

In [34]:

```python
X.boxplot()
```

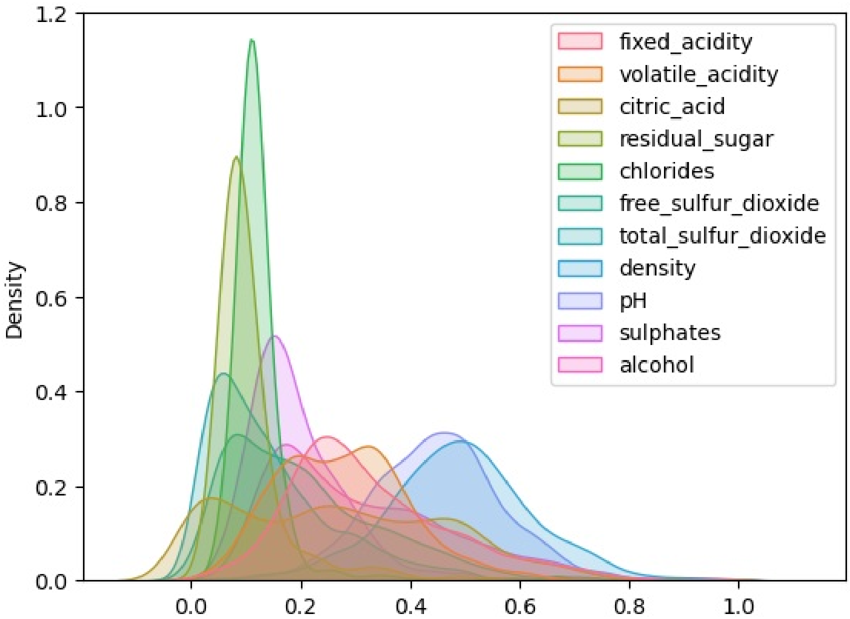
Out[34]: `<Axes: >`




In [35]:

```python
# KDE plot
sns.kdeplot(data=X, shade=True)
```

Out[35]: `<Axes: ylabel='Density'>`




In [36]:

```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
```

```
from sklearn.datasets import load_iris

# Load the Iris dataset



# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize the RandomForestClassifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the classifier
clf.fit(X_train, y_train)

# Make predictions on the test data
y_pred = clf.predict(X_test)

# Evaluate the model
print("Classification Report:")
print(classification_report(y_test, y_pred))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

         Bad       1.00      0.06      0.11        18
        Good       0.60      0.51      0.55        67
      Middle       0.88      0.94      0.91       395

    accuracy                           0.85       480
   macro avg       0.83      0.50      0.52       480
weighted avg       0.85      0.85      0.83       480

Confusion Matrix:
[[  1   0  17]
 [  0  34  33]
 [  0  23 372]]
```

In [37]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Initialize the RandomForestClassifier
lr= LogisticRegression( random_state=42)

# Train the classifier
lr.fit(X_train, y_train)

# Make predictions on the test data
y_pred = lr.predict(X_test)

# Evaluate the model
print("Classification Report:")
print(classification_report(y_test, y_pred))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

         Bad       0.00      0.00      0.00        18
        Good       0.59      0.19      0.29        67
      Middle       0.84      0.98      0.91       395

    accuracy                           0.83       480
   macro avg       0.48      0.39      0.40       480
weighted avg       0.78      0.83      0.79       480

Confusion Matrix:
[[  0   0  18]
 [  0  13  54]
 [  0   9 386]]
```

In [38]:
```
# Initialize the Support Vector Classifier
svc = SVC(kernel='linear', random_state=42)

# Train the classifier
svc.fit(X_train, y_train)

# Make predictions on the test data
y_pred = svc.predict(X_test)
```

```
# Evaluate the model
print("Classification Report:")
print(classification_report(y_test, y_pred))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

         Bad       0.00      0.00      0.00        18
        Good       0.00      0.00      0.00        67
      Middle       0.82      1.00      0.90       395

    accuracy                           0.82       480
   macro avg       0.27      0.33      0.30       480
weighted avg       0.68      0.82      0.74       480

Confusion Matrix:
[[  0   0  18]
 [  0   0  67]
 [  0   0 395]]
```

In [39]:

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix

# Introduce imbalance by removing some instances of one class
X = X[y != 2]
y = y[y != 2]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Standardize features by removing the mean and scaling to unit variance
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Initialize the Support Vector Classifier with class weight adjustment
svc = SVC(kernel='linear', class_weight='balanced', random_state=42)

# Train the classifier
svc.fit(X_train, y_train)

# Make predictions on the test data
y_pred = svc.predict(X_test)

# Evaluate the model
print("Classification Report:")
print(classification_report(y_test, y_pred))

print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

         Bad       0.14      0.89      0.24        18
        Good       0.40      0.84      0.54        67
      Middle       0.96      0.55      0.70       395

    accuracy                           0.60       480
   macro avg       0.50      0.76      0.50       480
weighted avg       0.85      0.60      0.66       480

Confusion Matrix:
[[ 16   0   2]
 [  4  56   7]
 [ 93  84 218]]
```

In [40]:

```
X = X[y != 2]
y = y[y != 2]
# Split data into training and test sets
train_X, test_X, train_y, test_y = train_test_split(X, y, random_state=42, test_size=0.3)

# Scale the data
scaler = StandardScaler()
X_train = scaler.fit_transform(train_X)
X_test = scaler.transform(test_X)

# Train the SVM classifier
```

```python
svc = SVC(kernel="linear", class_weight="balanced", random_state=42)
svc.fit(X_train, train_y)

# Predict quality based on user input
try:
    user_input = [float(x) for x in input("Enter values separated by spaces (e.g., '1.5 2.5'): ").split()]
    user_input_scaled = scaler.transform([user_input])  # Scale the input using the same scaler
    prediction = svc.predict(user_input_scaled)
    print("Predicted quality is:", prediction[0])
except Exception as e:
    print("Error:", e)
```

```
Enter values separated by spaces (e.g., '1.5 2.5'): 7.4 0.700   0.00    1.9     0.076   11.0    34.0    0.99780
3.51    0.56 9.4
Predicted quality is: Middle
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: