Red Wine Quality Prediction Project

Project Description

The dataset is related to red and white variants of the Portuguese "Vinho Verde" wine. Due to privacy and logistic issues, only physicochemical (inputs) and sensory (the output) variables are available (e.g. there is no data about grape types, wine brand, wine selling price, etc.).

This dataset can be viewed as classification task. The classes are ordered and not balanced (e.g. there are many more normal wines than excellent or poor ones). Also, we are not sure if all input variables are relevant. So it could be interesting to test feature selection methods.

Attribute Information

Input variables (based on physicochemical tests): 1 - fixed acidity 2 - volatile acidity 3 - citric acid 4 - residual sugar 5 - chlorides 6 - free sulfur dioxide 7 - total sulfur dioxide 8 - density 9 - pH 10 - sulphates 11 - alcohol Output variable (based on sensory data): 12 - quality (score between 0 and 10)

In [52]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_m
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import MinMaxScaler
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
```

In [3]:
```
df = pd.read_csv('winequality-red (1).csv')
```

In [4]:
```
df.head()
```

Out[4]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4

In [5]:
```
df.shape
```

Out[5]:
```
(1599, 12)
```

In [6]:
```
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 [7]:
```
# checking for missing vlaues
df.isnull().sum()
```

Out[7]:
```
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
```

There are no missing values.

In [8]:
```
df['quality'].value_counts()
```

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

In [9]:
```
df.describe()
```

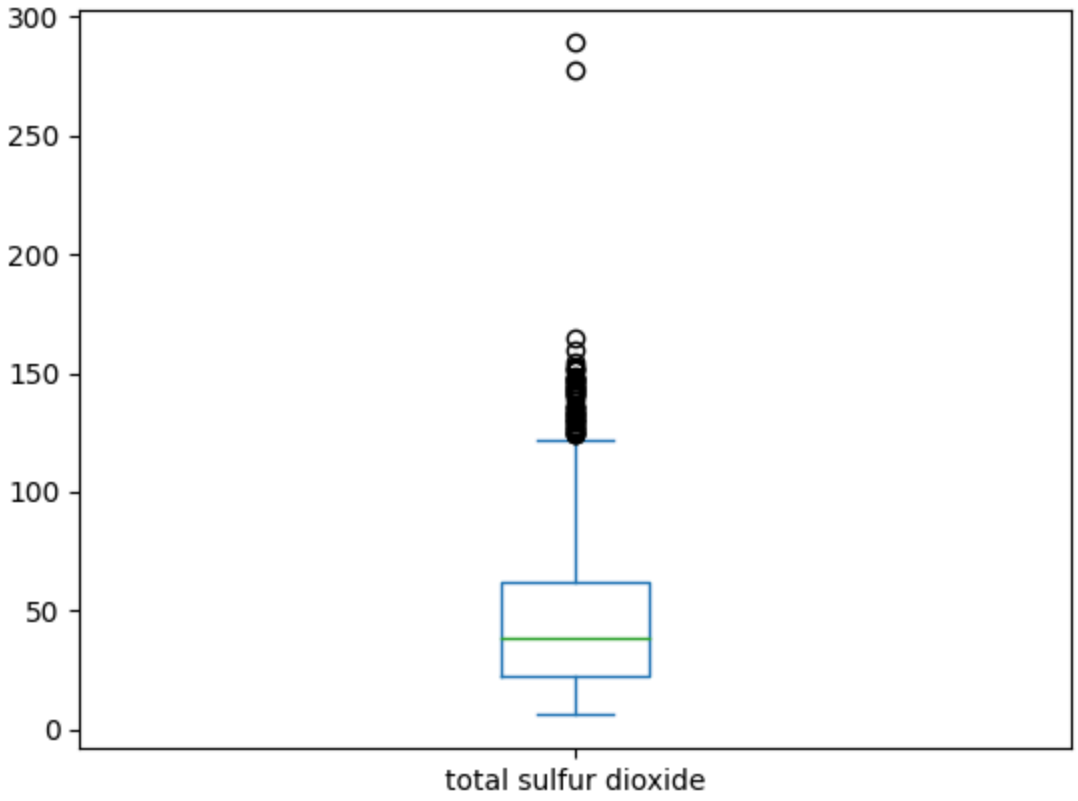
Out[9]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulf dioxi
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.0000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.4677
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.8953
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.0000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.0000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.0000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.0000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.0000

Visualization and plots

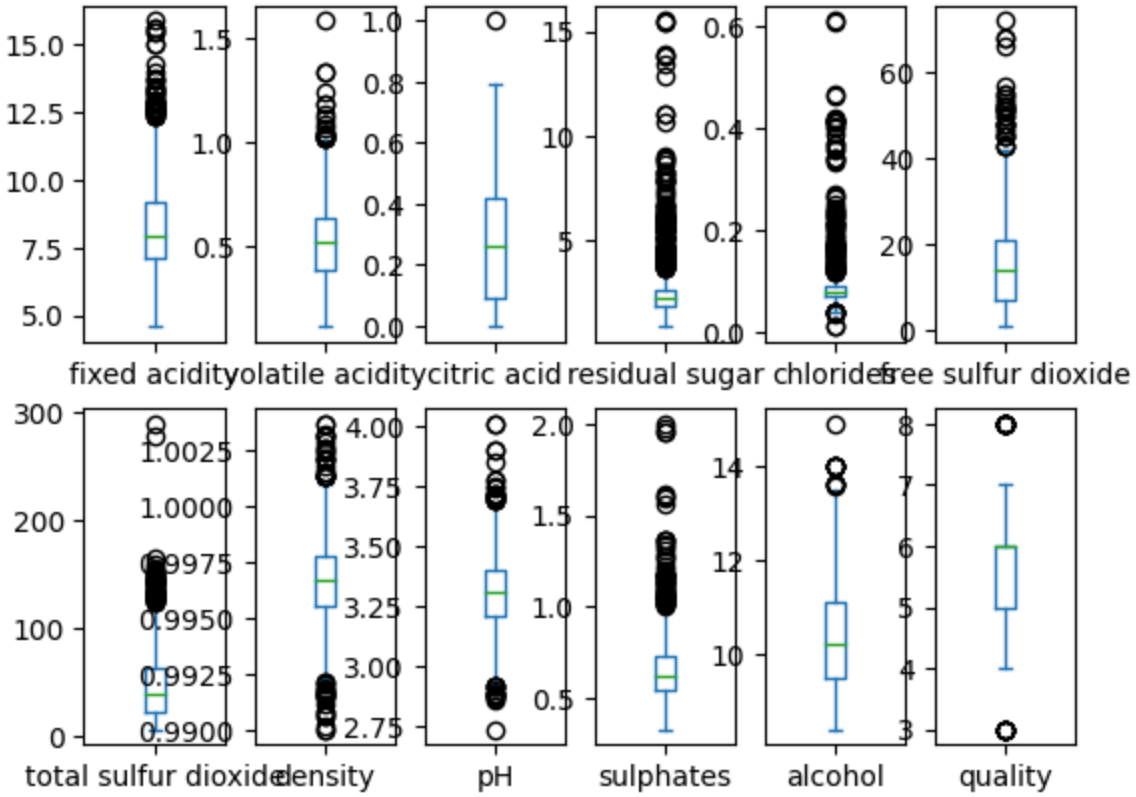
In [10]:
```
df['total sulfur dioxide'].plot.box()
```

Out[10]: <Axes: >



In [11]:
```
df.plot(kind='box',subplots=True, layout=(2,6))
```

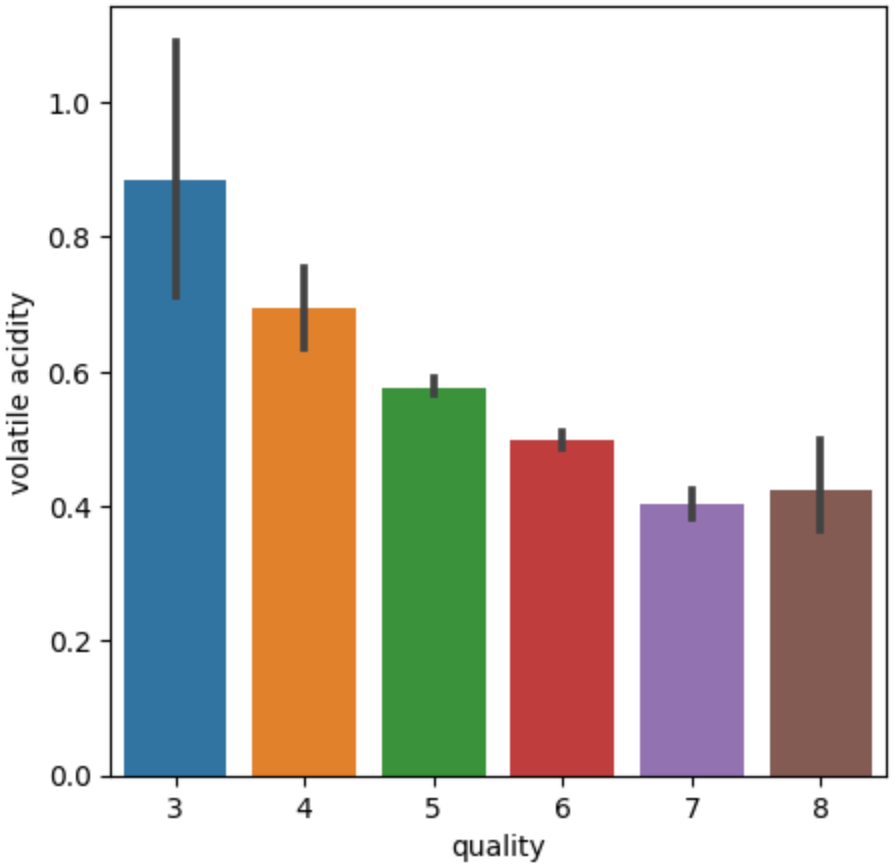
Out[11]:
```
fixed acidity             Axes(0.125,0.53;0.110714x0.35)
volatile acidity       Axes(0.257857,0.53;0.110714x0.35)
citric acid            Axes(0.390714,0.53;0.110714x0.35)
residual sugar         Axes(0.523571,0.53;0.110714x0.35)
chlorides              Axes(0.656429,0.53;0.110714x0.35)
free sulfur dioxide    Axes(0.789286,0.53;0.110714x0.35)
total sulfur dioxide      Axes(0.125,0.11;0.110714x0.35)
density                Axes(0.257857,0.11;0.110714x0.35)
pH                     Axes(0.390714,0.11;0.110714x0.35)
sulphates              Axes(0.523571,0.11;0.110714x0.35)
alcohol                Axes(0.656429,0.11;0.110714x0.35)
quality                Axes(0.789286,0.11;0.110714x0.35)
dtype: object
```



There is no Outlier in data.

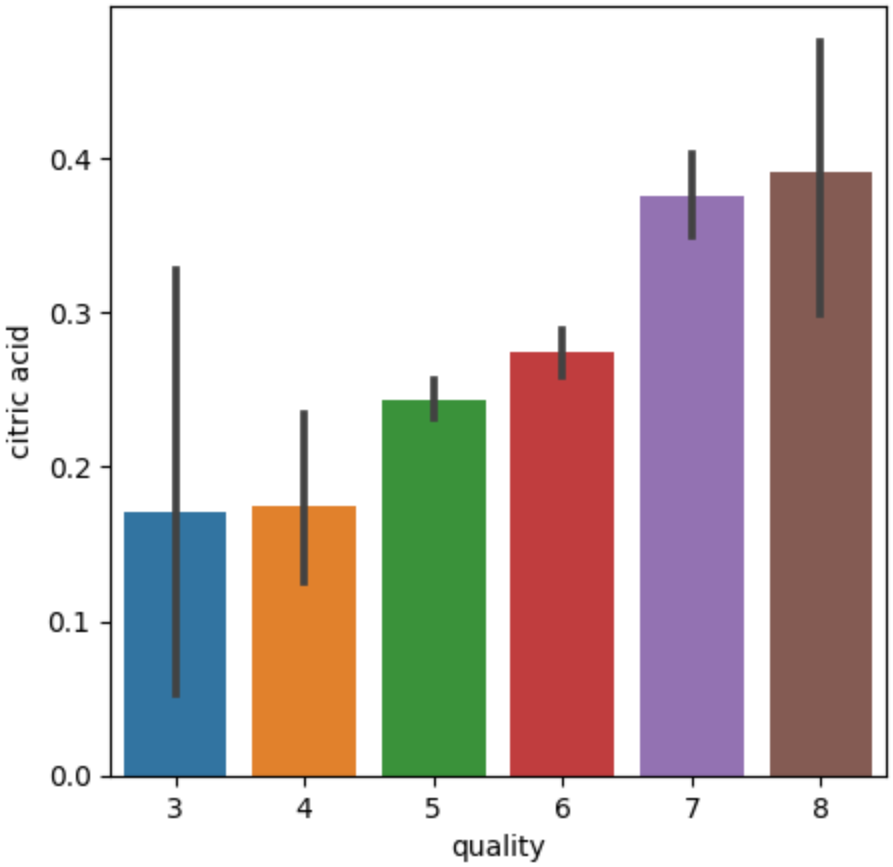
In [12]:
```
fig=plt.figure(figsize=(5,5))
sns.barplot(x='quality',y='volatile acidity',data=df)
```

Out[12]: <Axes: xlabel='quality', ylabel='volatile acidity'>



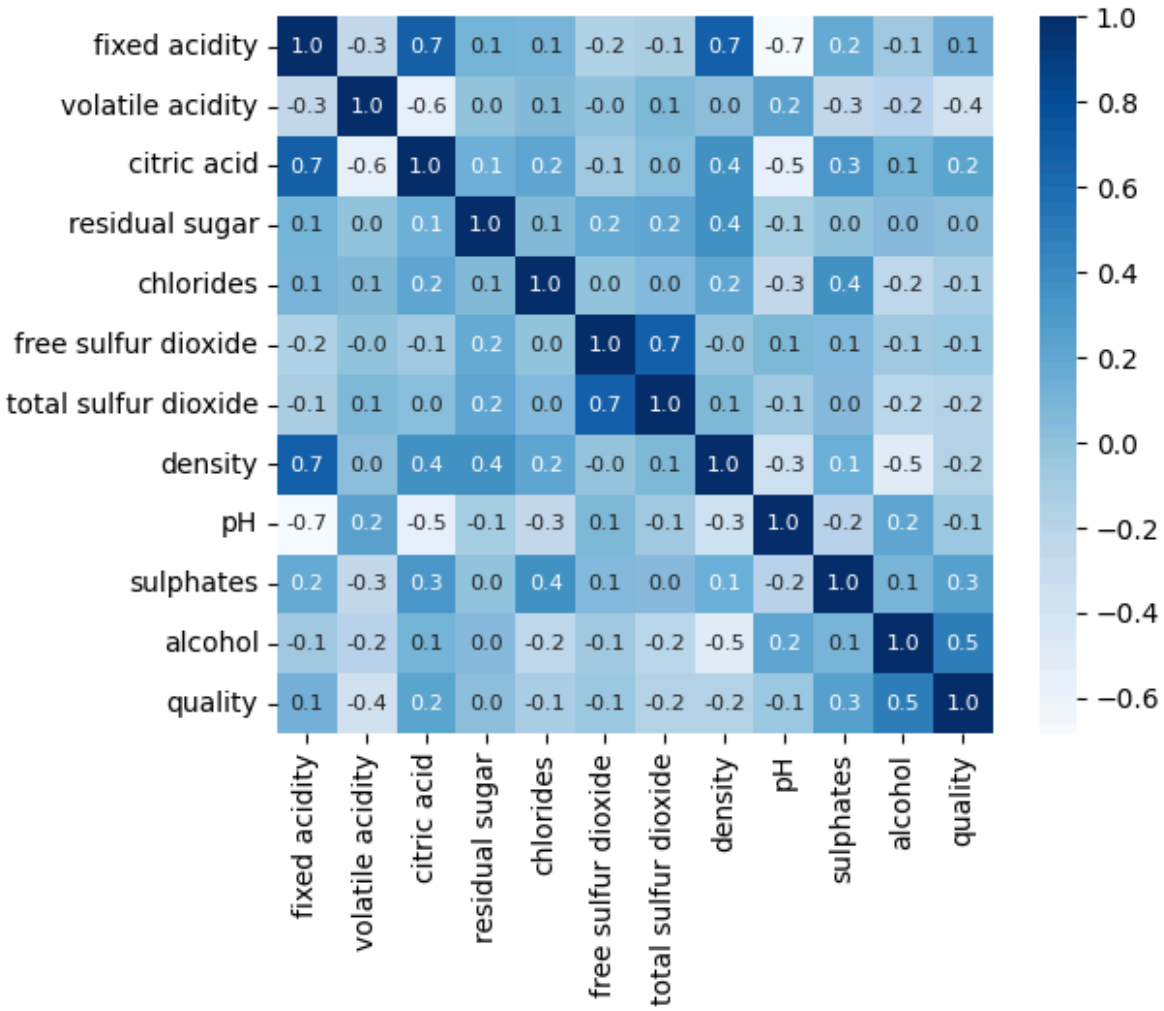
In [13]:
```
fig=plt.figure(figsize=(5,5))
sns.barplot(x='quality',y='citric acid',data=df)
```

Out[13]: <Axes: xlabel='quality', ylabel='citric acid'>



In [14]:
```
correlation=df.corr()
sns.heatmap(correlation, cbar=True,square=True,  fmt = '.1f', annot = True, an
```

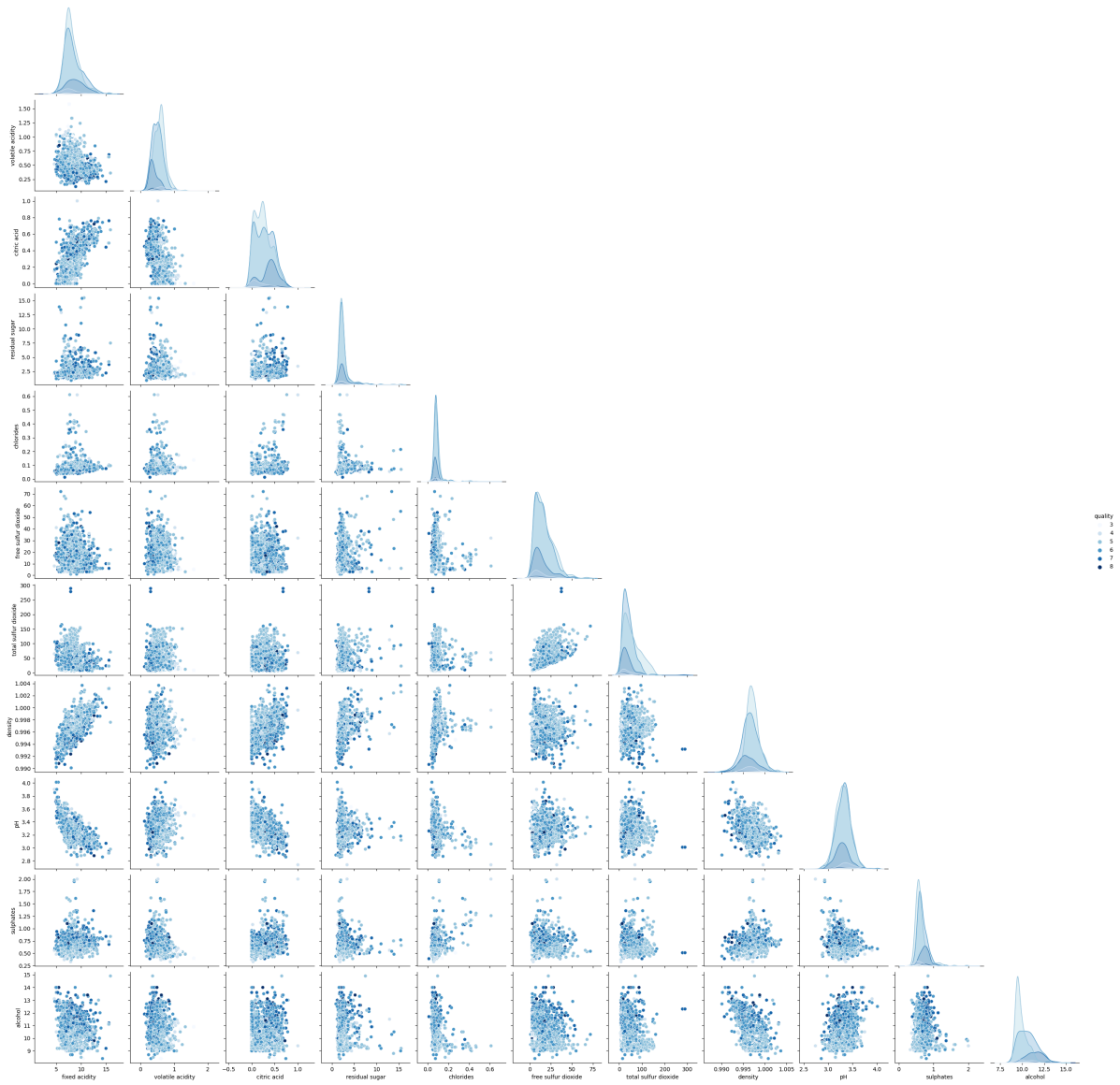
Out[14]: <Axes: >



Best Correlations are between : citeic_acid & flex_acidity ---> 0.7 density & flex_acidity ---> 0.7 total_sulfor_dioxide & free_sulfor_dioxide ---> 0.7

In [23]:
```
sns.pairplot(df, hue='quality', corner = True, palette='Blues')
```

Out[23]:
```
<seaborn.axisgrid.PairGrid at 0x1e09c0f6510>
```



In [28]:
```
X=df.drop("quality",axis=1)
Y=df['quality'].apply(lambda x:1 if x>=7 else  0)
```

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

In [29]:
```
Y.value_counts()
```

Out[29]:
```
0    1382
1     217
Name: quality, dtype: int64
```

Normalization

In [30]:
```
scaler = MinMaxScaler(feature_range=(0, 1)).fit_transform(X)
X = pd.DataFrame(scaler, columns=X.columns)
```

In [31]:
```
print(X)
```

```
      fixed acidity  volatile acidity  citric acid  residual sugar  chlorides
\
0          0.247788          0.397260         0.00        0.068493   0.106845
1          0.283186          0.520548         0.00        0.116438   0.143573
2          0.283186          0.438356         0.04        0.095890   0.133556
3          0.584071          0.109589         0.56        0.068493   0.105175
4          0.247788          0.397260         0.00        0.068493   0.106845
...             ...               ...          ...             ...        ...
1594       0.141593          0.328767         0.08        0.075342   0.130217
1595       0.115044          0.294521         0.10        0.089041   0.083472
1596       0.150442          0.267123         0.13        0.095890   0.106845
1597       0.115044          0.359589         0.12        0.075342   0.105175
1598       0.123894          0.130137         0.47        0.184932   0.091820

      free sulfur dioxide  total sulfur dioxide   density        pH  \
0                0.140845              0.098940  0.567548  0.606299
1                0.338028              0.215548  0.494126  0.362205
2                0.197183              0.169611  0.508811  0.409449
3                0.225352              0.190813  0.582232  0.330709
4                0.140845              0.098940  0.567548  0.606299
...                   ...                   ...       ...       ...
1594             0.436620              0.134276  0.354626  0.559055
1595             0.535211              0.159011  0.370778  0.614173
1596             0.394366              0.120141  0.416300  0.535433
1597             0.436620              0.134276  0.396476  0.653543
1598             0.239437              0.127208  0.397944  0.511811

      sulphates   alcohol
0      0.137725  0.153846
1      0.209581  0.215385
2      0.191617  0.215385
3      0.149701  0.215385
4      0.137725  0.153846
...         ...       ...
1594   0.149701  0.323077
1595   0.257485  0.430769
1596   0.251497  0.400000
1597   0.227545  0.276923
1598   0.197605  0.400000

[1599 rows x 11 columns]
```

In [32]:
```
X.describe()
```

Out[32]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulf dioxi
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.0000
mean	0.329171	0.279329	0.270976	0.112247	0.125988	0.209506	0.1429
std	0.154079	0.122644	0.194801	0.096570	0.078573	0.147326	0.1162
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
25%	0.221239	0.184932	0.090000	0.068493	0.096828	0.084507	0.0565
50%	0.292035	0.273973	0.260000	0.089041	0.111853	0.183099	0.1130
75%	0.407080	0.356164	0.420000	0.116438	0.130217	0.281690	0.1978
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0000

In [34]:
```
# Splite Dataframe
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.25, rand
```

RandomForestClassifier

In [40]:
```
model = RandomForestClassifier()
model.fit(X_train, y_train)
```

Out[40]:
```
RandomForestClassifier
RandomForestClassifier()
```

In [43]:
```
# accuracy on test data
X_test_prediction = model.predict(X_test)
# test_data_accuracy = accuracy_score(X_test_prediction, Y_test)
test_data_accuracy = accuracy_score( y_test,X_test_prediction)
print('Accuracy : ', test_data_accuracy)
```

```
Accuracy :  0.9225
```

Building a Predictive System

In [44]:
```
input_data = (7.5,0.5,0.36,6.1,0.071,17.0,102.0,0.9978,3.35,0.8,10.5)

# changing the input data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the data as we are predicting the label for only one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

prediction = model.predict(input_data_reshaped)
print(prediction)

if (prediction[0]==1):
  print('Good Quality Wine')
else:
  print('Bad Quality Wine')
```

```
[0]
Bad Quality Wine

C:\Users\99Minds-1\AppData\Local\Programs\Python\Python311\Lib\site-packages
\sklearn\base.py:439: UserWarning: X does not have valid feature names, but R
andomForestClassifier was fitted with feature names
  warnings.warn(
```

In [46]:
```
confuion_matrix = confusion_matrix(y_test, X_test_prediction)
```

In [48]:
```
confuion_matrix
```

Out[48]:
```
array([[342,  13],
       [ 18,  27]], dtype=int64)
```

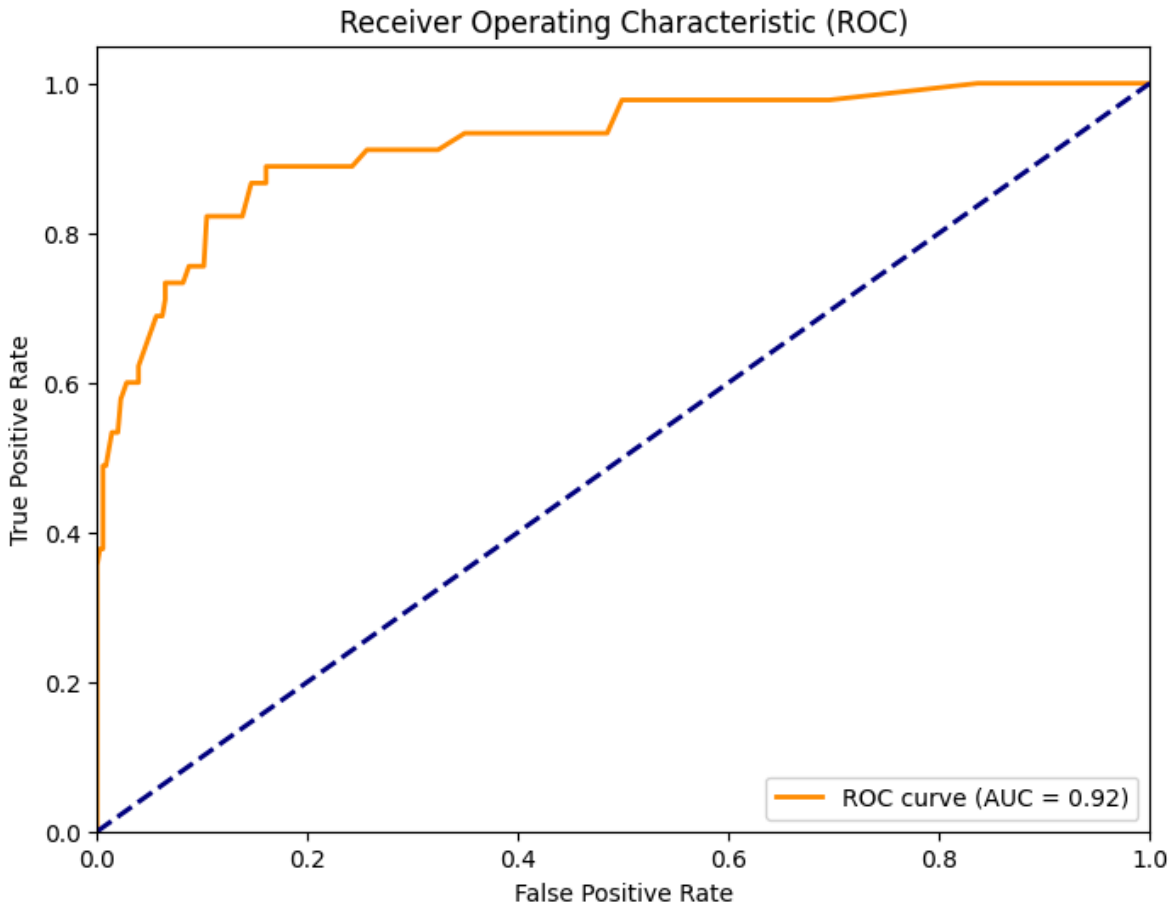
In [50]:
```
 classification_report(y_test, X_test_prediction)
```

Out[50]:
```
'              precision    recall  f1-score   support\n\n           0
0.95      0.96      0.96       355\n           1       0.68      0.60      0.
64        45\n\n    accuracy                           0.92       400\n   mac
ro avg       0.81      0.78      0.80       400\nweighted avg       0.92
0.92      0.92       400\n'
```

ROC and AUC value

In [55]:
```
# Calculate ROC curve and AUC score
fpr, tpr, _ = roc_curve(y_test, model.predict_proba(X_test)[:, 1])
roc_auc = auc(fpr, tpr)

# Plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
plt.show()
```



Grid Search

In [56]:
```
from sklearn.model_selection import GridSearchCV
# Define a grid of hyperparameters to search
param_grid = {
    'n_estimators': [50, 100, 150, 200],
    'max_depth': [None, 10, 20, 30]
}

# Create a GridSearchCV object
grid_search = GridSearchCV(estimator=rf, param_grid=param_grid, cv=5, scoring=
grid_search.fit(X_train, y_train)

# Print the best parameters and the corresponding accuracy
print("Best Parameters:", grid_search.best_params_)
print("Best Accuracy:", grid_search.best_score_)
```

```
Best Parameters: {'max_depth': 20, 'n_estimators': 50}
Best Accuracy: 0.902433751743375
```

Thankyou

In [ ]:
```

```