Red Wine Quality Prediction

Problem Statement:

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)

What might be an interesting thing to do, is to set an arbitrary cutoff for your dependent variable (wine quality) at e.g. 7 or higher getting classified as 'good/1' and the remainder as 'not good/0'. This allows you to practice with hyper parameter tuning on e.g. decision tree algorithms looking at the ROC curve and the AUC value.

In [1]:
```
#Importing Required libraries
```

```
import pandas as pd
import numpy as np
import seaborn  as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler,LabelEncoder
from sklearn.metrics import confusion_matrix,classification_report
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
df=pd.read_csv('winequality_red.csv')
df.head()
```

Out[2]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	qu
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 [3]:
```
# check the row and columns
df.shape
```

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

In [4]:
```
#check information about dataset
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 [5]:
```
#check null values or not
df.isnull().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
```

Here no found any null values

In [6]:
```
df.columns
```

Out[6]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

Descriptive Statistics

In [7]:
```
df.describe()
```

Out[7]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	1
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	

Data visualization

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

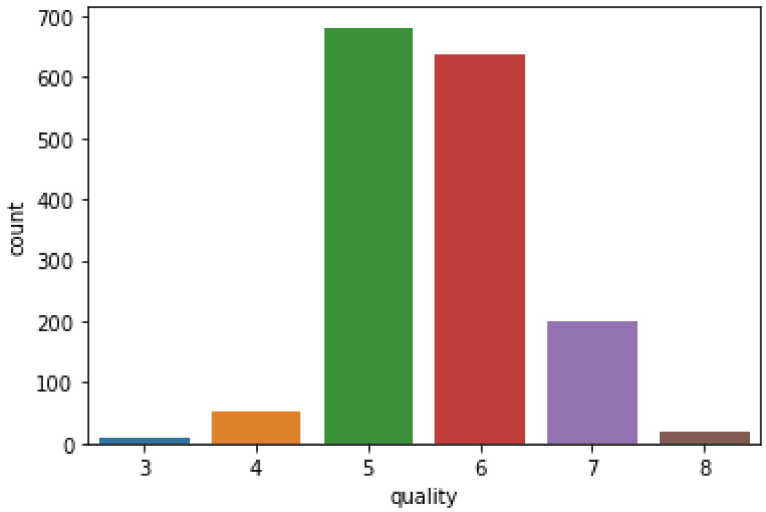
Out[8]:
```
array([5, 6, 7, 4, 8, 3], dtype=int64)
```

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

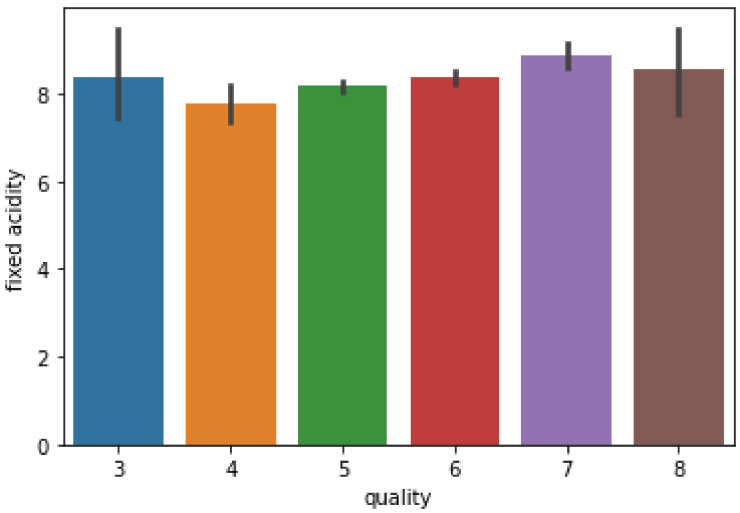
Out[9]:
```
1599
```

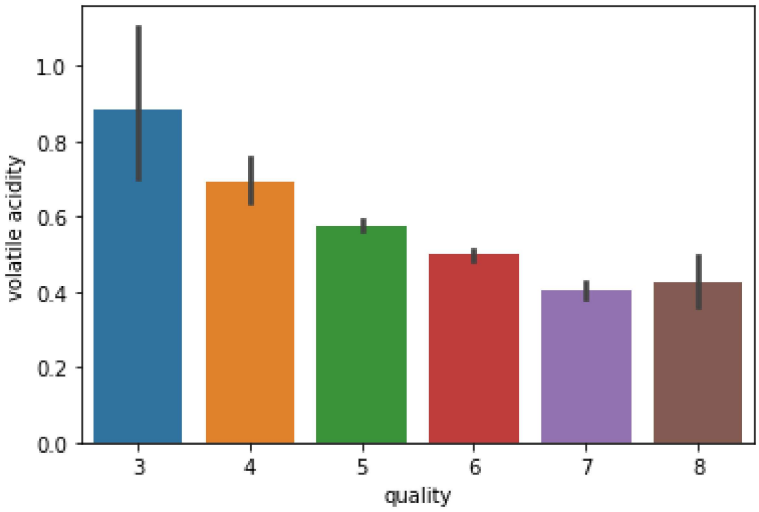
In [10]:
```
sns.countplot(x='quality',data=df)
df['quality'].value_counts()
```

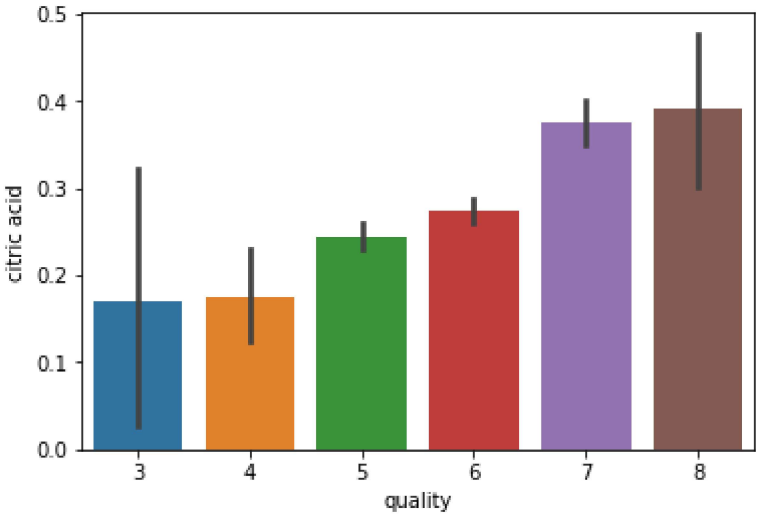
Out[10]:
```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

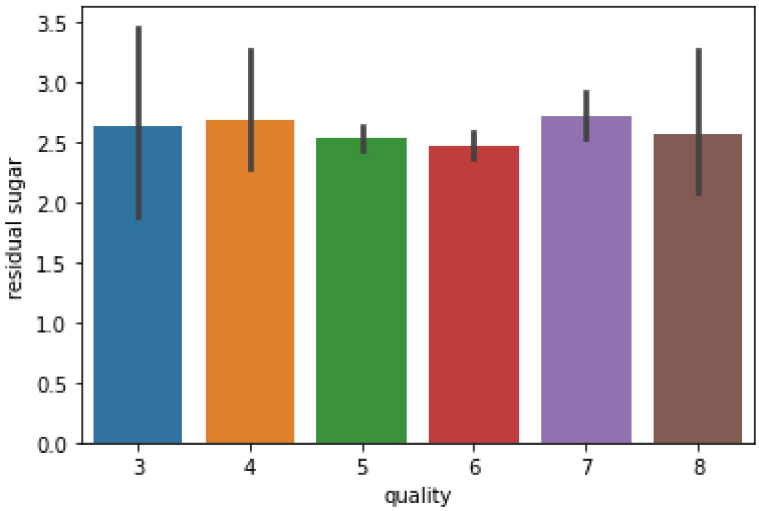


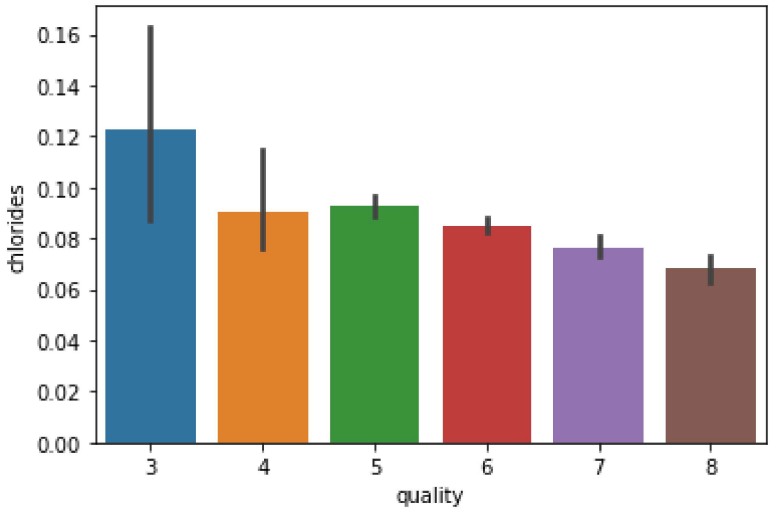
In [11]:
```
df1=df.select_dtypes([np.int,np.float])
for i,col in enumerate(df1.columns):
    plt.figure(i)
    sns.barplot(x='quality',y=col,data=df1)
```

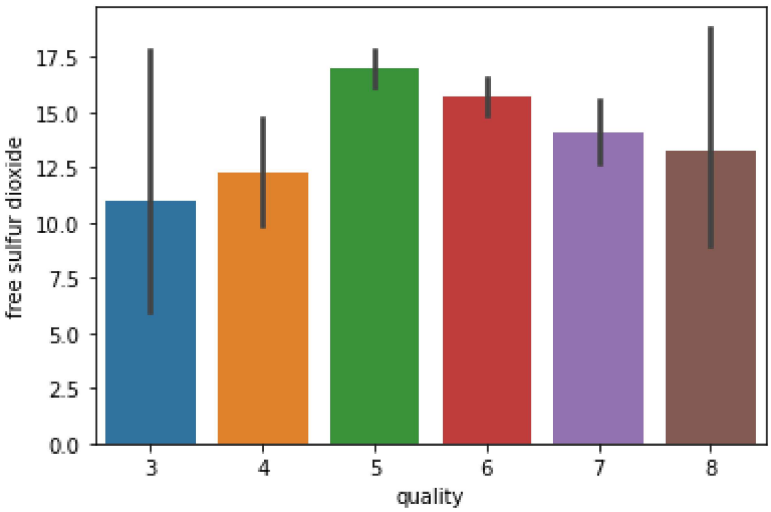


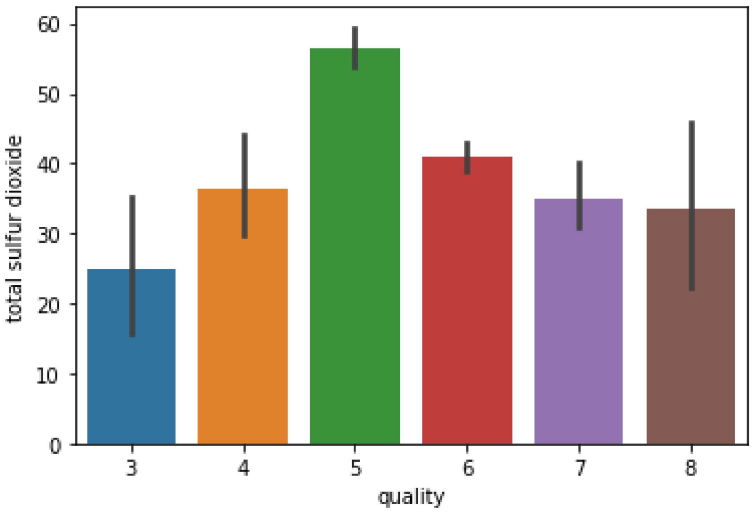

volatile acidity
1.0
0.8
0.6
0.4
0.2
0.0
3
4
5
6
7
8
quality

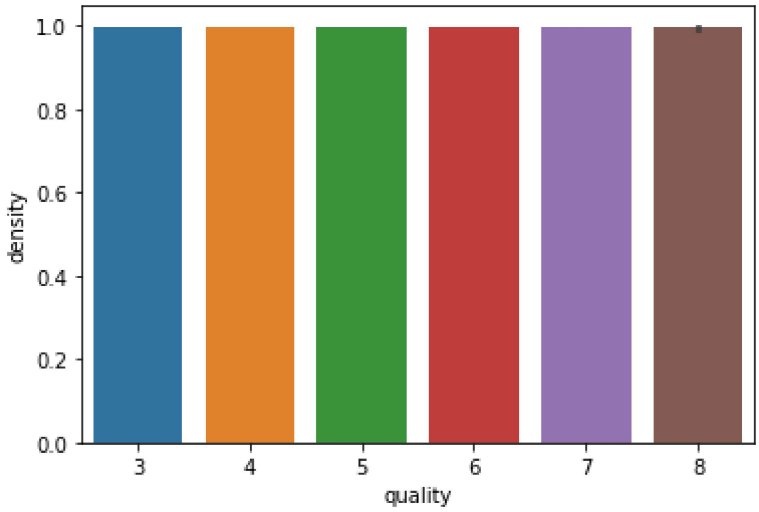

citric acid
0.5
0.4
0.3
0.2
0.1
0.0
3
4
5
6
7
8
quality

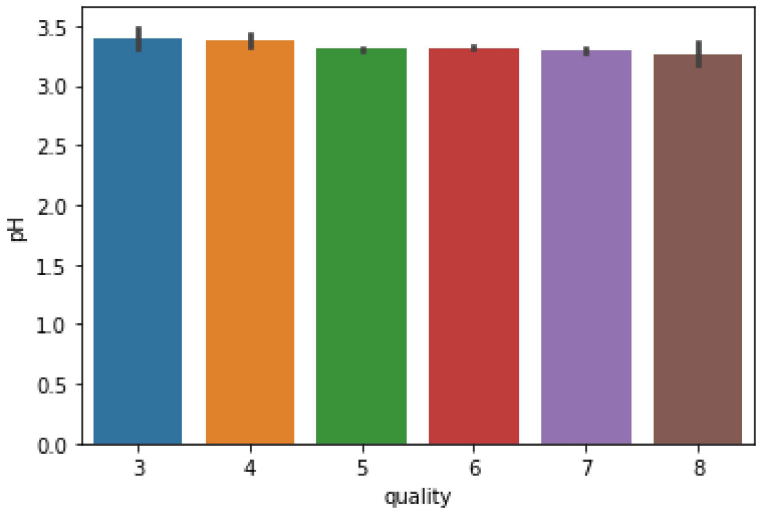

residual sugar
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
3
4
5
6
7
8
quality

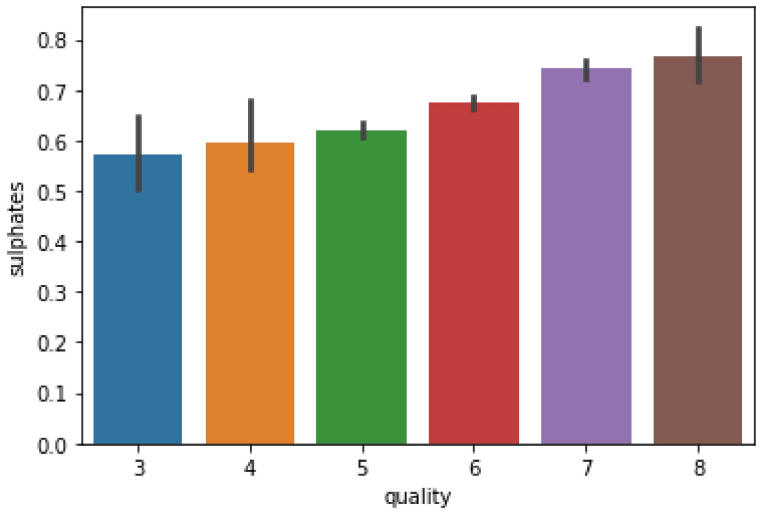

chlorides
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
3
4
5
6
7
8
quality

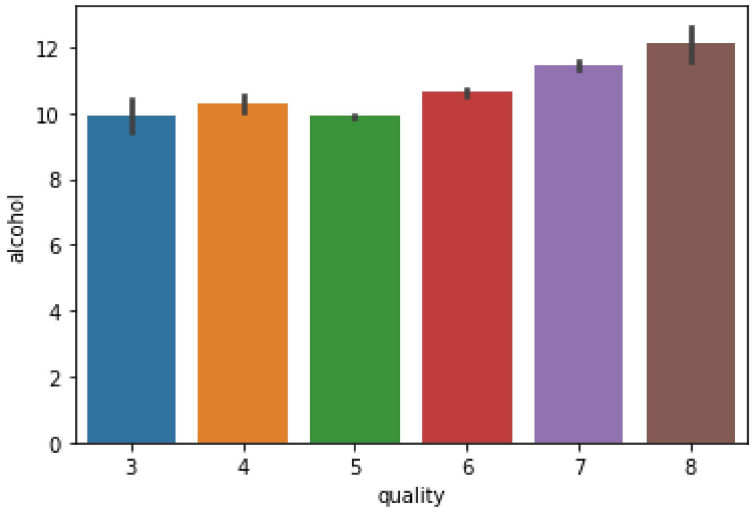

free sulfur dioxide
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
3
4
5
6
7
8
quality

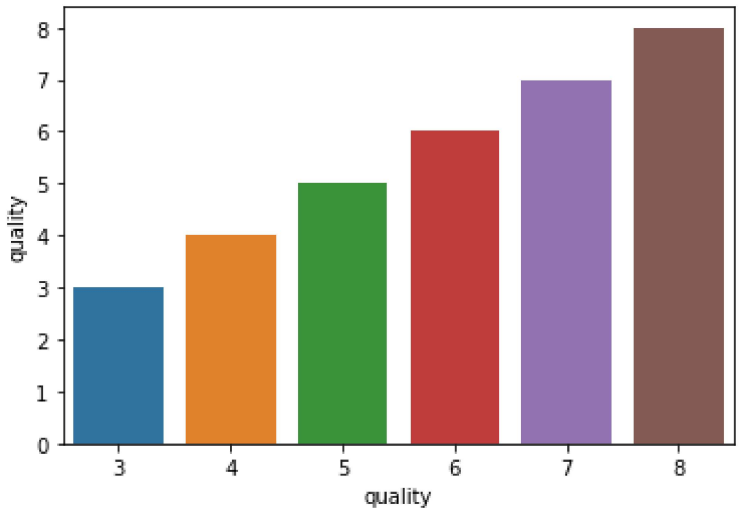

total sulfur dioxide
60
50
40
30
20
10
0
3
4
5
6
7
8
quality


density
1.0
0.8
0.6
0.4
0.2
0.0
3
4
5
6
7
8
quality


pH
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
3
4
5
6
7
8
quality


sulphates
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
3
4
5
6
7
8
quality



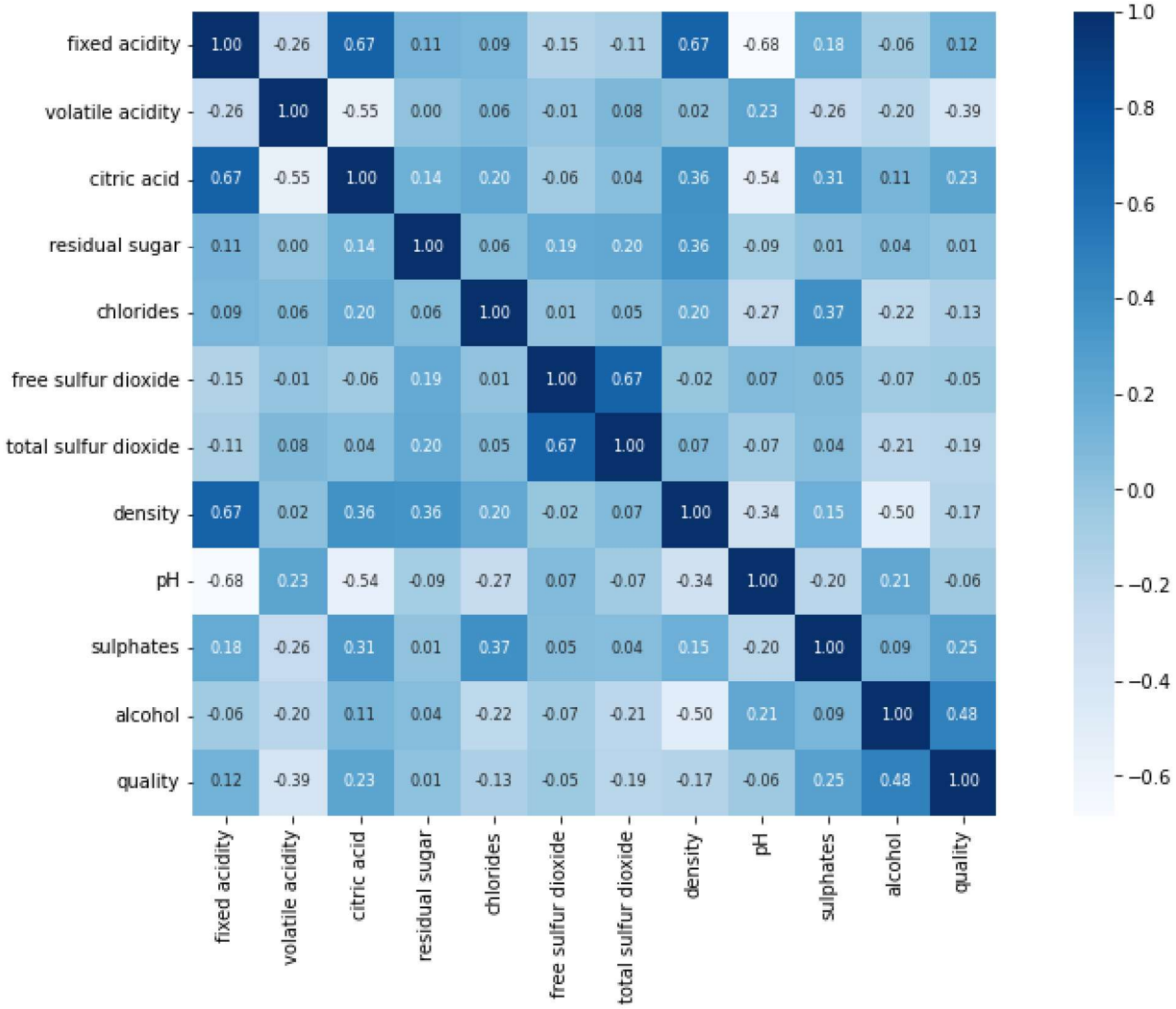


Correlattion

1. Positive Correlation
2. Negative Correlation

In [12]:
```
plt.figure(figsize=(15,8))
sns.heatmap(df1.corr(),cbar=True,square=True,fmt='.2f',annot=True,annot_kws={'size':8}
```

Out[12]:
```
<AxesSubplot:>
```



Creating Classification bins

In [13]:
```
bins=(2,6.5,8)
group_name=['bad','good']
df1['quality']=pd.cut(df1['quality'], bins=bins,labels=group_name)
```

Classification into ondes and zeros using LabelEncoder

In [14]:
```
#Assigning a label to our quality variable
label_qulity=LabelEncoder()

#changing our daraframe to reflect our new label
df1['quality']=label_qulity.fit_transform(df1['quality'])
```

In [15]:
```
df1.head(10)
```

Out[15]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	qu
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	
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	9.4	
6	7.9	0.60	0.06	1.6	0.069	15.0	59.0	0.9964	3.30	0.46	9.4	
7	7.3	0.65	0.00	1.2	0.065	15.0	21.0	0.9946	3.39	0.47	10.0	
8	7.8	0.58	0.02	2.0	0.073	9.0	18.0	0.9968	3.36	0.57	9.5	
9	7.5	0.50	0.36	6.1	0.071	17.0	102.0	0.9978	3.35	0.80	10.5	

Setting the dependent and independent variable

In [16]:
```
y=df1.quality
x=df1.drop('quality',axis=1)
```

Train & Test Split

In [17]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y, test_size=0.2,random_state=0)
```

In [18]:
```
print(y.shape,y_train.shape,y_test.shape)
```

```
(1599,) (1279,) (320,)
```

Here total rows = 1599

give traning data rows= 1279

give test data rows= 320

Scaling the data to take account of variations in mean and Standard deviations

In [19]:
```
#Feature scaling
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.transform(x_test)
```

Model Training

Machine Learning Models

In [51]:
```
#Create a fuction within many machine learning Models
def models(x_train,y_train):

    #using Logistic Regression Algorithm to the Training set
    from sklearn.linear_model import LogisticRegression
    lr=LogisticRegression()
    lr.fit(x_train,y_train)

    #using KNeighborsClassifier Method of neighbors class to use Nearest Neighbor algo
    from sklearn.neighbors import KNeighborsClassifier
    knn=KNeighborsClassifier(n_neighbors=5,metric='minkowski',p=2)
    knn.fit(x_train,y_train)

    #using SVC method of svm class use Support Vector Machine Algorithm
    from sklearn.svm import SVC
    svc=SVC(kernel='linear',random_state=0)
    svc.fit(x_train,y_train)

    #using GaussinNB method of navie_bayes class to use Naive Bayes Algorithm
    from sklearn.naive_bayes import GaussianNB
    gnb=GaussianNB()
    gnb.fit(x_train,y_train)

    #using DecisionTreeClassifier of tree class to use Decision Tree Classifier algori
    from sklearn.tree import DecisionTreeClassifier
    dtc=DecisionTreeClassifier()
    dtc.fit(x_train,y_train)

    #using RandomForestClassifier method of ensemble class to use Random Forest Classi
    from sklearn.ensemble import RandomForestClassifier
    rfc=RandomForestClassifier(n_estimators=11,criterion='entropy',random_state=0)
    rfc.fit(x_train,y_train)

    #print model accuracy on the training data.
    print('[0] Logistic Regression Training Accuracy : ',lr.score(x_train,y_train))
    print('[1] K Nearest Neighbor Training Accuracy : ',knn.score(x_train,y_train))
    print('[2] Support Vector Machine(Linear Classifier) Training Accuracy : ',svc.sc
    print('[3] Gussian Naive Bayes Training Accuracy : ',gnb.score(x_train,y_train))
    print('[4] Decision Tree Classifier Training Accuracy : ',dtc.score(x_train,y_trai
    print('[5] Random Forest Classifier Training Accuracy : ',rfc.score(x_train,y_trai

    return lr,knn,svc,gnb,dtc,rfc
```

Evaluating Performance on Training Sets

In [52]:
```
#Get and train all of the models
model=models(x_train,y_train)
```

```
[0] Logistic Regression Training Accuracy :  0.8733385457388585
[1] K Nearest Neighbor Training Accuracy :  0.8983580922595777
[2] Support Vector Machine(Linear Classifier) Training Accuracy :  0.8537920250195465
[3] Gussian Naive Bayes Training Accuracy :  0.8358092259577795
[4] Decision Tree Classifier Training Accuracy :  1.0
[5] Random Forest Classifier Training Accuracy :  0.9945269741985927
```

Evaluating Performance on Training Sets

In [53]:
```
from sklearn.metrics import confusion_matrix
for i in range(len(model)):
    cm=confusion_matrix(y_test, model[i].predict(x_test))
    #extracting TN,FP,FN,TP
    TN,FP,FN,TP = confusion_matrix(y_test, model[i].predict(x_test)).ravel()
    print(cm)
    print('model[{}] Testing Accuracy="{} !"'.format(i,(TP + TN) / (TP + TN + FN + FP)
    print() #Print a new line
```

```
[[277  13]
 [ 18  12]]
model[0] Testing Accuracy="0.903125 !"

[[276  14]
 [ 13  17]]
model[1] Testing Accuracy="0.915625 !"

[[290   0]
 [ 30   0]]
model[2] Testing Accuracy="0.90625 !"

[[240  50]
 [  6  24]]
model[3] Testing Accuracy="0.825 !"

[[264  26]
 [  8  22]]
model[4] Testing Accuracy="0.89375 !"

[[276  14]
 [ 11  19]]
model[5] Testing Accuracy="0.921875 !"
```

Random Forest Classifier Algorithm

In [54]:
```
 #using RandomForestClassifier method of ensemble class to use Random Forest Classific
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier(n_estimators=11,criterion='entropy',random_state=0)
rfc.fit(x_train,y_train)
```

Out[54]:

RandomForestClassifier

```
RandomForestClassifier(criterion='entropy', n_estimators=11, random_state=0)
```

Accuracy Score

In [55]:
```
accuracy_score=rfc.score(x_test,y_test)
accuracy_score
```

Out[55]:
```
0.921875
```

Building a Predictive System

In [56]:
```
input_data=(6.6,0.725,0.2,7.8,0.073,29.0,79.0,0.9977,3.29,0.54,9.2)

#Changing the input dta to a numpy array
input_data_as_np = np.asarray(input_data)

#Reshape the data as we are prediction the label for only one instance
input_data_reshape=input_data_as_np.reshape(1,-1)

prediction=rfc.predict(input_data_reshape)
print(prediction)

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

```
[0]
Bad Quality Wine
```

Model Saving

In [57]:
```
import pickle
filename='Red Wine Quality Prediction.pkl'
pickle.dump(rfc, open(filename,'wb'))
```

conclusion

In [58]:
```
a=np.array(y_test)
predicted=np.array(rfc.predict(x_test))
df.com=pd.DataFrame({"original":a,"predicted":predicted},index=range(len(a)))
df.com
```

Out[58]:

	original	predicted
0	0	0
1	0	0
2	1	1
3	0	0
4	0	0
...	...	...
315	0	0
316	0	0
317	0	0
318	0	0
319	0	0

320 rows × 2 columns

- Rondom forest the best Model to give best prediction