MACHINE LEARNING CLASSIFICATION ASSIGNMENT REPORT & CODE - SOMESH GAUR

In [158]:
```
### THIS CODE IS REQUIRED AS I AM WORKING ON IBM WATSON PLATFORM ####

# @hidden_cell
# The project token is an authorization token that is used to access project r
esources like data sources, connections, and used by platform APIs.
from project_lib import Project
project = Project(project_id='00018ec1-6194-427c-8ce8-67d29662e2a8', project_a
ccess_token='p-7e008f5b3a3bbf2681434775bc94dd48755a4fec')
pc = project.project_context
# Write  "Wine_Quality_Data.csv" to working directory
open('/home/wsuser/work/Wine_Quality_Data.csv','wb').write(project.get_file('W
ine_Quality_Data.csv').read())
```

Out[158]: 464197

In [160]:
```
import pandas as pd
import numpy as np
#import os -> remove if not required
import seaborn as sns
import matplotlib.pyplot as plt
#import plotly.express as px -> remove if not required
wdata=pd.read_csv('/home/wsuser/work/Wine_Quality_Data.csv',sep=',')
```

A) Main objective of the analysis and the benefits that analysis provides to the business or stakeholders of this data.

MAIN OBJECTIVE: The main objective of this excercise is to build and evalaute various Classification models to predict whether a particular wine is “good quality” or not based on their features. While it may be interesting to understand the interpretibility of the model, but from practical and usefulness stand-point , I would like to focus on predictability of model in this excercise.

BENEFITS: While many people like wines but very few people can really differentiate between quality of wines. It may be of interest and importance to individual, restarants and wineries to know what makes a good wine. But such prediction by a human might vary from person to person and require highly skilled tasters.It will be very useful if we can develop machine learning models to predict a good quality wine based on its physical/chemical composition.

B) Brief description of the data set, a summary of its attributes, and an outline of objective of analysis.

BRIEF DESCRIPTION OF DATASET: I am using Wine_Quality_Data.csv which a popular datset used for ML models. This data set contains various chemical properties of wine, such as acidity, sugar, pH, and alcohol. It also contains a quality metric (3-9, with highest being better) and a color (red or white).

This dataset is originally based on two datasets that are 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.).

SUMMARY OF ATTRIBUTES :

There are 13 features and 6497 rows in the dataset. 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 , 12- Color (Physical quality) , 13 - Quality (score between 0 and 10) - Output variable (based on sensory data)

BRIEF DESCRIPTION of 12 variables and 1 output variable (quality).

Fixed Acidity: are non-volatile acids that do not evaporate readily

Volatile Acidity: are high acetic acid in wine which leads to an unpleasant vinegar taste

Citric Acid: acts as a preservative to increase acidity. When in small quantities, adds freshness and flavor to wines

Residual Sugar: is the amount of sugar remaining after fermentation stops. The key is to have a perfect balance between sweetness and sourness. It is important to note that wines > 45g/ltrs are sweet

Chlorides: the amount of salt in the wine

Free Sulfur Dioxide: it prevents microbial growth and the oxidation of wine

Total Sulfur Dioxide: is the amount of free + bound forms of SO2

Density: sweeter wines have a higher density

pH: describes the level of acidity on a scale of 0–14. Most wines are always between 3–4 on the pH scale

Alcohol: available in small quantities in wines makes the drinkers sociable

Sulphates: a wine additive that contributes to SO2 levels and acts as an antimicrobial and antioxidant

Quality: which is the output variable/predictor

Color : Color of wine (Red/White)

Datatypes: There are float64 :11 , int64 :1 and object: 1

OBJECTIVE OF ANALYSIS : The main aim of this excercise is to predict quality of wine based on physiochemical features. I am going to develop various classification models and evaluate which of them is best in predicting the quality of wine based on the above 12 features.

In [4]:
```
print(wdata.dtypes)
```

```
fixed_acidity           float64
volatile_acidity        float64
citric_acid             float64
residual_sugar          float64
chlorides               float64
free_sulfur_dioxide     float64
total_sulfur_dioxide    float64
density                 float64
pH                      float64
sulphates               float64
alcohol                 float64
quality                   int64
color                    object
dtype: object
```

In [4]:
```
wdata.dtypes.value_counts()
```

Out[4]:
```
float64    11
int64       1
object      1
dtype: int64
```

In [3]:
```
wdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed_acidity         6497 non-null   float64
 1   volatile_acidity      6497 non-null   float64
 2   citric_acid           6497 non-null   float64
 3   residual_sugar        6497 non-null   float64
 4   chlorides             6497 non-null   float64
 5   free_sulfur_dioxide   6497 non-null   float64
 6   total_sulfur_dioxide  6497 non-null   float64
 7   density               6497 non-null   float64
 8   pH                    6497 non-null   float64
 9   sulphates             6497 non-null   float64
 10  alcohol               6497 non-null   float64
 11  quality               6497 non-null   int64  
 12  color                 6497 non-null   object 
dtypes: float64(11), int64(1), object(1)
memory usage: 660.0+ KB
```

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

Out[5]:
```
(6497, 13)
```

C) Brief summary of data exploration and actions taken for data cleaning and feature engineering.

In [9]:
```
### WRTIE SUMMARY ###

#DATA EXPLORATION :

#DATA CLEANING :

#FEATURE ENGINEERING :
```

C-1 : DATA EXPLORATION

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

Out[4]:

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	total_s
0	7.4	0.70	0.00	1.9	0.076	11.0	
1	7.8	0.88	0.00	2.6	0.098	25.0	
2	7.8	0.76	0.04	2.3	0.092	15.0	
3	11.2	0.28	0.56	1.9	0.075	17.0	
4	7.4	0.70	0.00	1.9	0.076	11.0	

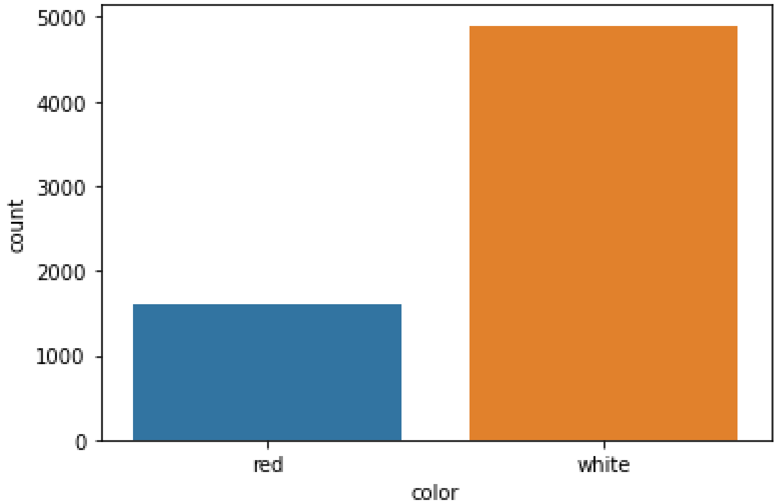
In [7]:
```
wdata.describe().T
```

Out[7]:

	count	mean	std	min	25%	50%	75%	
fixed_acidity	6497.0	7.215307	1.296434	3.80000	6.40000	7.00000	7.70000	15.
volatile_acidity	6497.0	0.339666	0.164636	0.08000	0.23000	0.29000	0.40000	1.
citric_acid	6497.0	0.318633	0.145318	0.00000	0.25000	0.31000	0.39000	1.
residual_sugar	6497.0	5.443235	4.757804	0.60000	1.80000	3.00000	8.10000	65.
chlorides	6497.0	0.056034	0.035034	0.00900	0.03800	0.04700	0.06500	0.
free_sulfur_dioxide	6497.0	30.525319	17.749400	1.00000	17.00000	29.00000	41.00000	289.
total_sulfur_dioxide	6497.0	115.744574	56.521855	6.00000	77.00000	118.00000	156.00000	440.
density	6497.0	0.994697	0.002999	0.98711	0.99234	0.99489	0.99699	1.
pH	6497.0	3.218501	0.160787	2.72000	3.11000	3.21000	3.32000	4.
sulphates	6497.0	0.531268	0.148806	0.22000	0.43000	0.51000	0.60000	2.
alcohol	6497.0	10.491801	1.192712	8.00000	9.50000	10.30000	11.30000	14.
quality	6497.0	5.818378	0.873255	3.00000	5.00000	6.00000	6.00000	9.

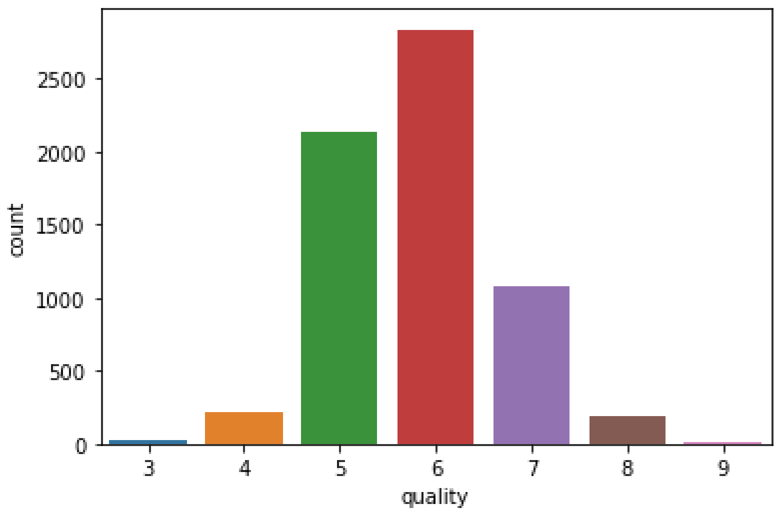
In [8]:
```
# Data distribution of wine color/type.
sns.countplot(x='color',data=wdata)
wdata['color'].value_counts()
### Clearly, we have data imbalance here with much more data with respect to r
ed wine ###
```

Out[8]:
```
white    4898
red      1599
Name: color, dtype: int64
```



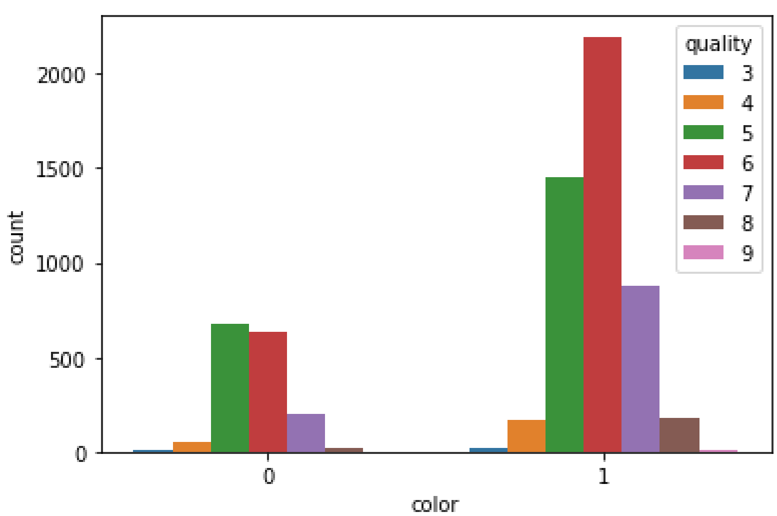
In [9]:
```
# Let's check data distribution of wine quality, which is our target variable
sns.countplot(x='quality',data=wdata)
wdata.quality.value_counts()
### Clearly, we have data imbalance here with much more data with respect to Q
uality=5,6,7 ###
```

Out[9]:
```
6    2836
5    2138
7    1079
4     216
8     193
3      30
9       5
Name: quality, dtype: int64
```

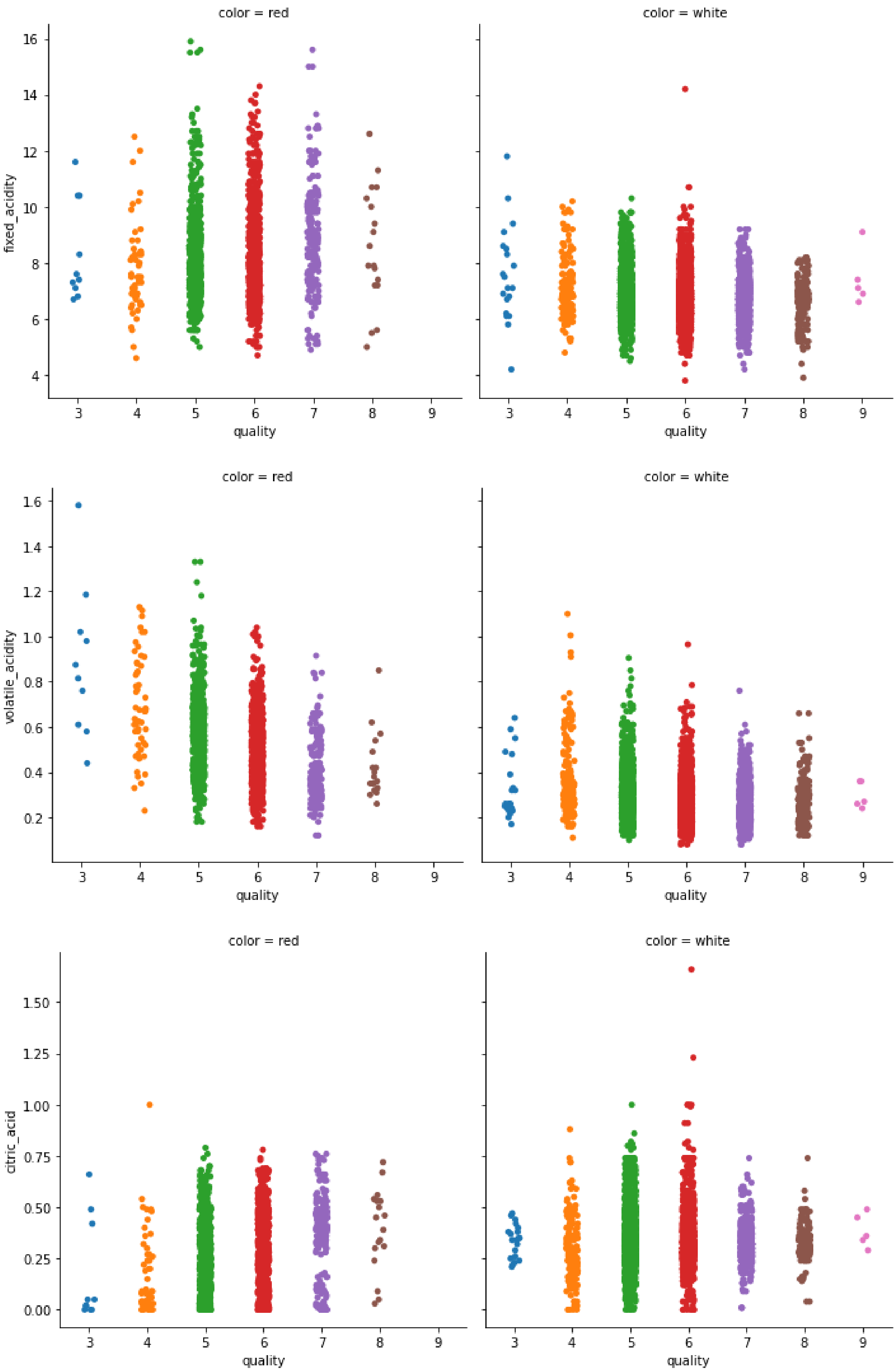


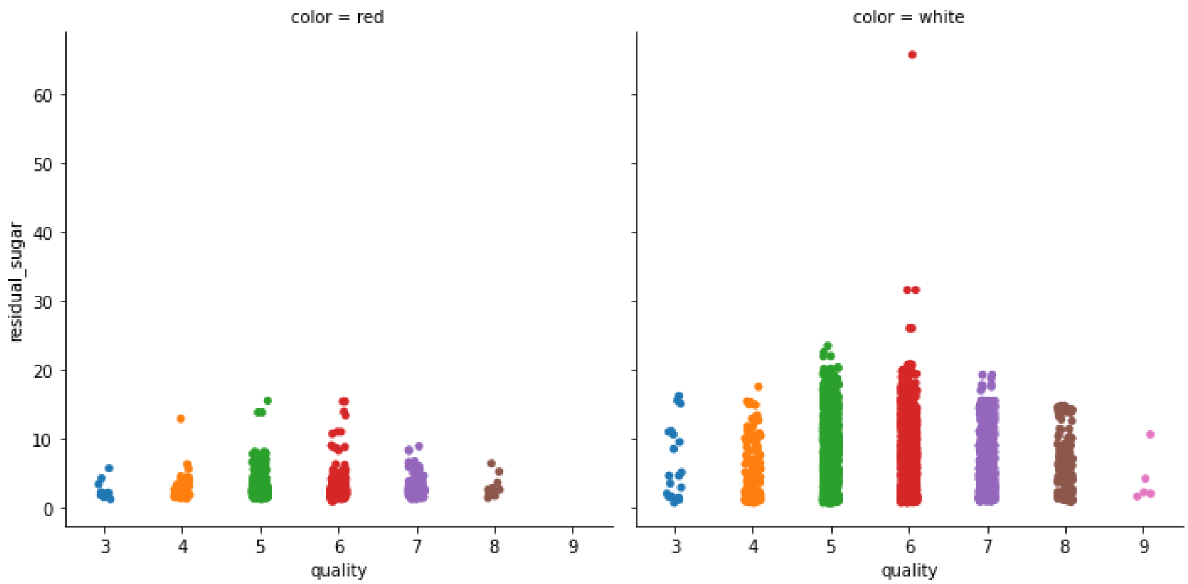
In [15]:
```
plt.figure(figsize=(6,4))
sns.countplot(x='color', hue='quality',data=wdata)
```

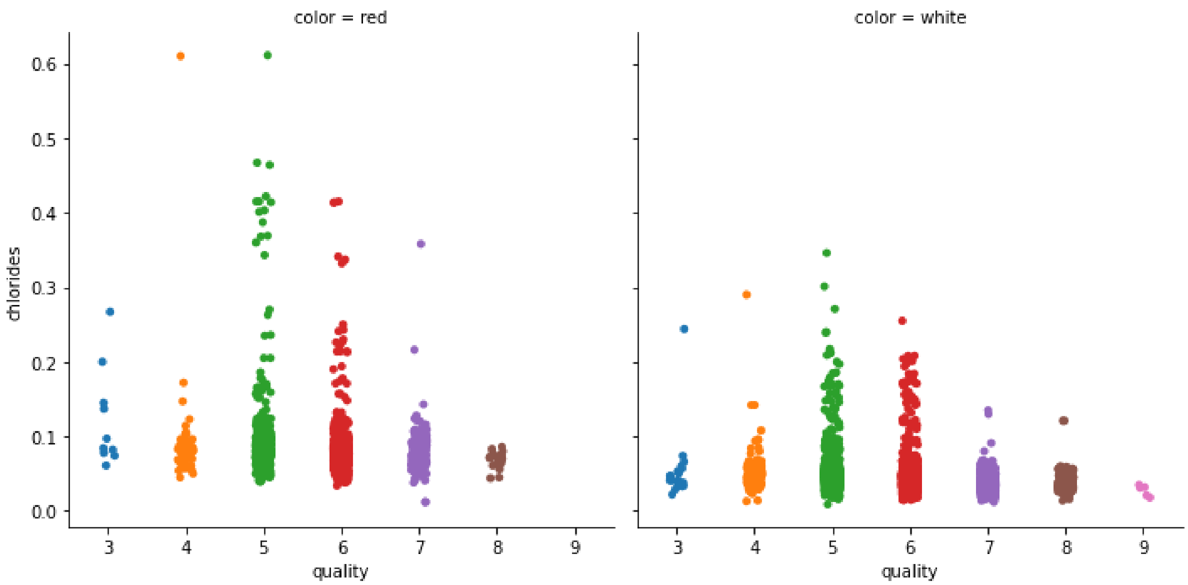
Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x7fad6021a610>

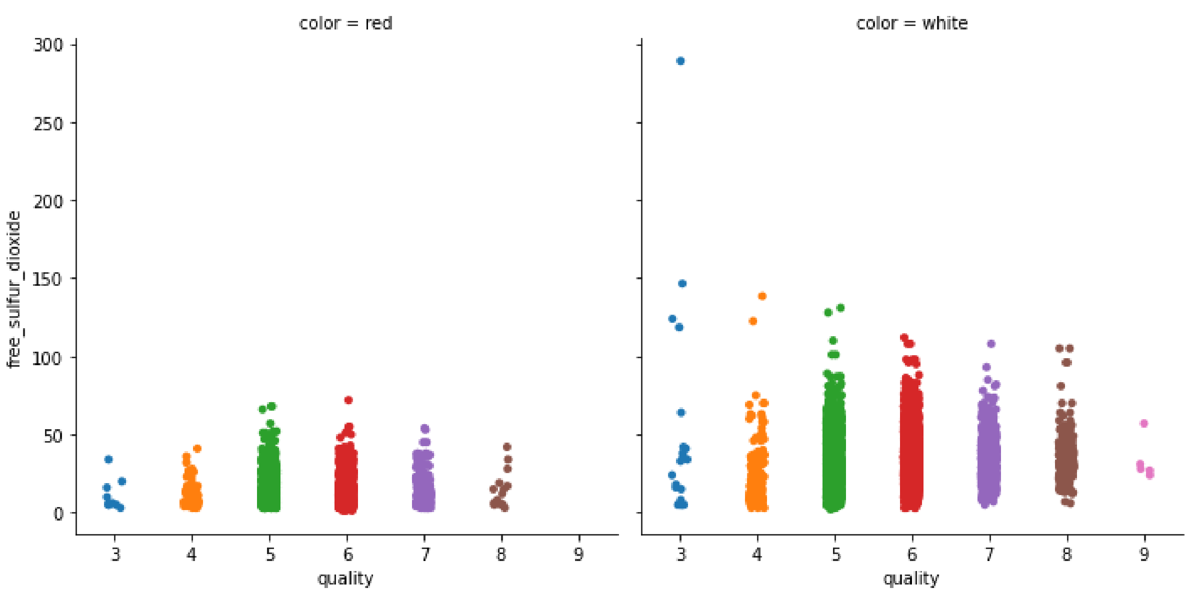


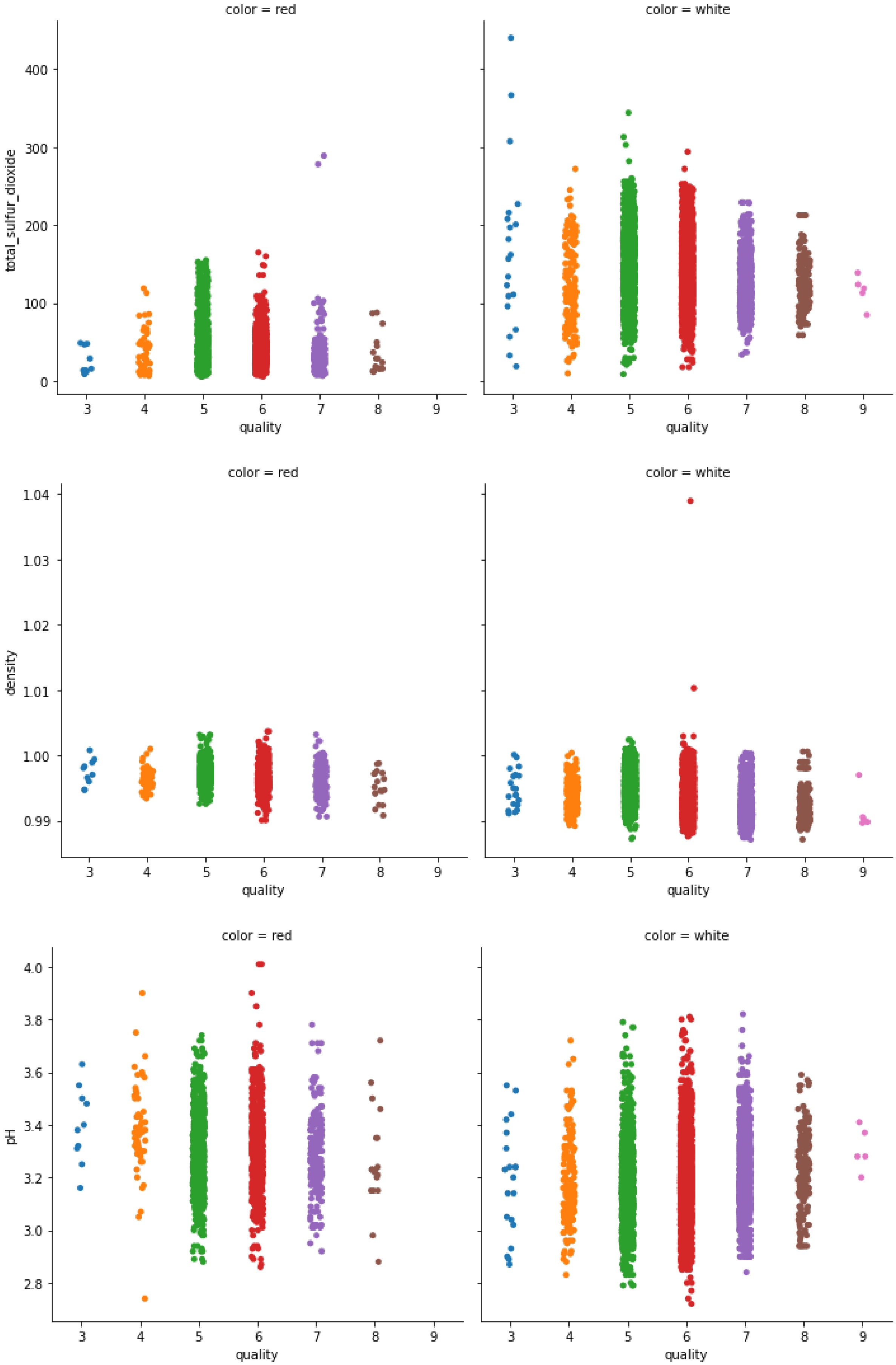
In [54]:
```
# numerical columns
numeric_data = [feature for feature in wdata.columns if wdata[feature].dtypes
!='O' and feature not in 'quality']

# Let's see the distribution of data with Quality ( on red and white wine )
# Also, we can develop some sense of 'Suspected outliers'

for feature in numeric_data:
    sns.catplot(x='quality',y=feature,col='color',data=wdata)
    plt.show()
```

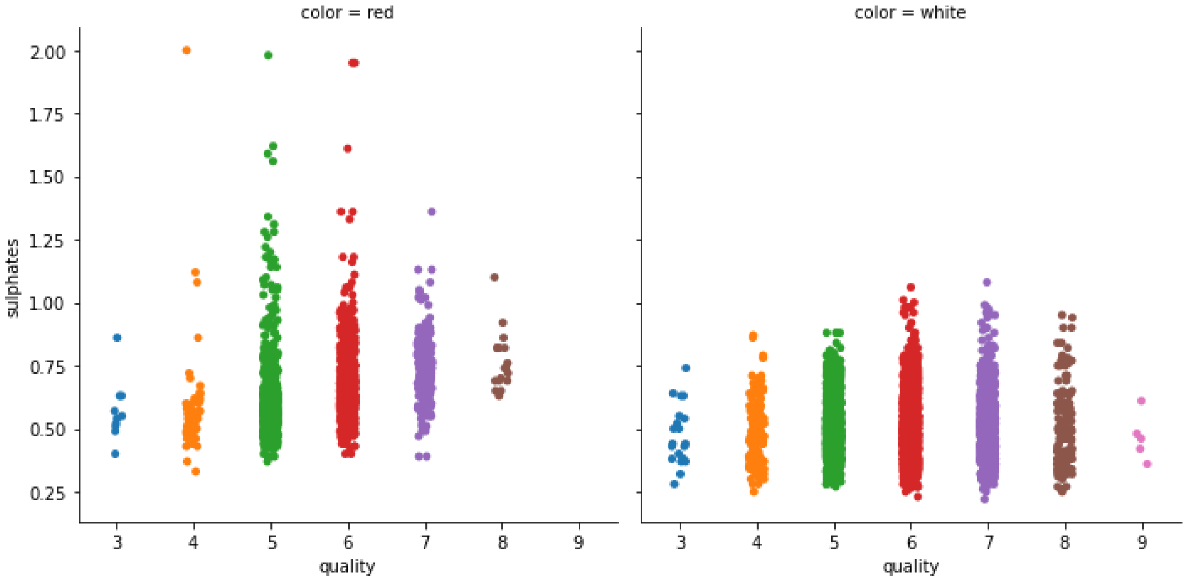

color = red
color = white
fixed_acidity
16
14
12
10
8
6
4
3
4
5
6
7
8
9
quality
color = red
color = white
volatile_acidity
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
quality
color = red
color = white
citric_acid
1.50
1.25
1.00
0.75
0.50
0.25
0.00
quality

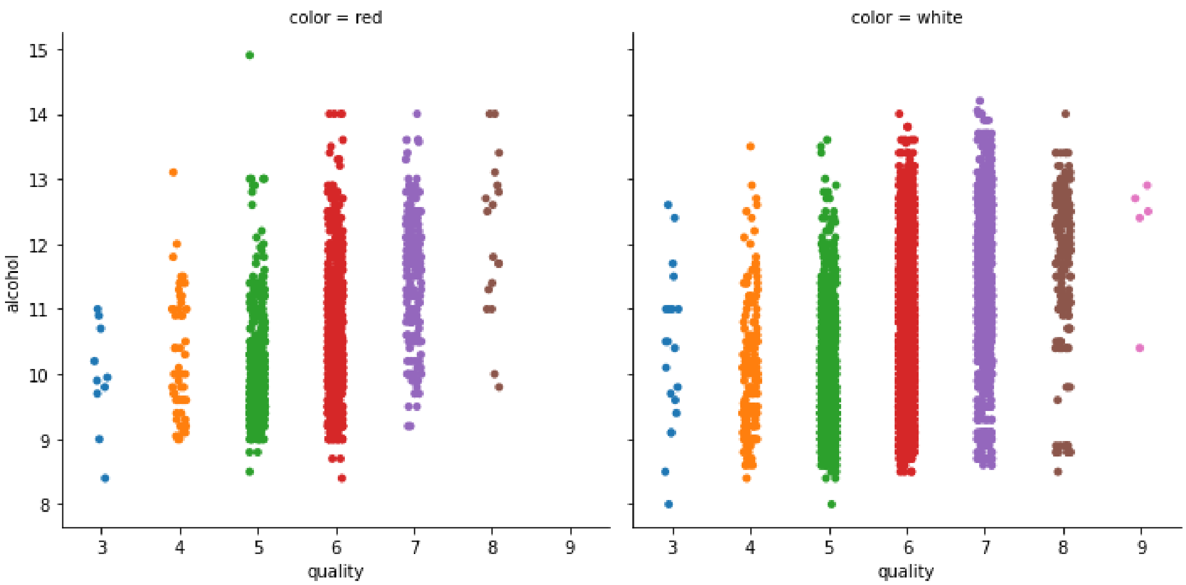

color = red
color = white
residual_sugar
quality
0
10
20
30
40
50
60
3
4
5
6
7
8
9


color = red
color = white
chlorides
quality
0.0
0.1
0.2
0.3
0.4
0.5
0.6
3
4
5
6
7
8
9

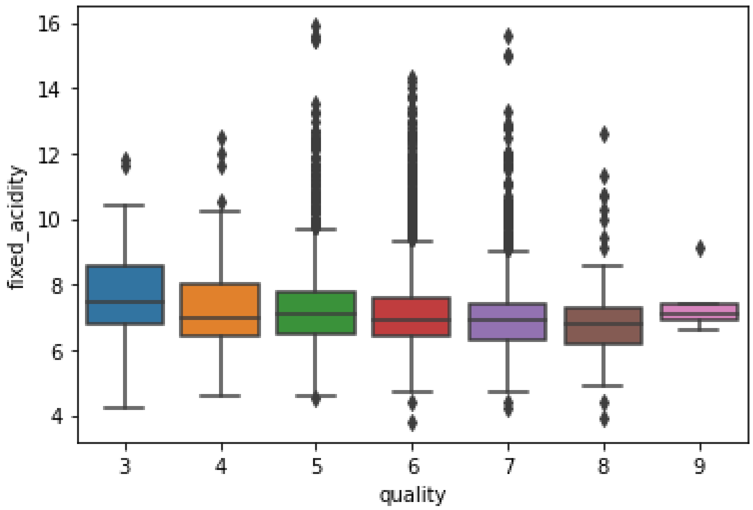

color = red
color = white
free_sulfur_dioxide
quality
0
50
100
150
200
250
300
3
4
5
6
7
8
9

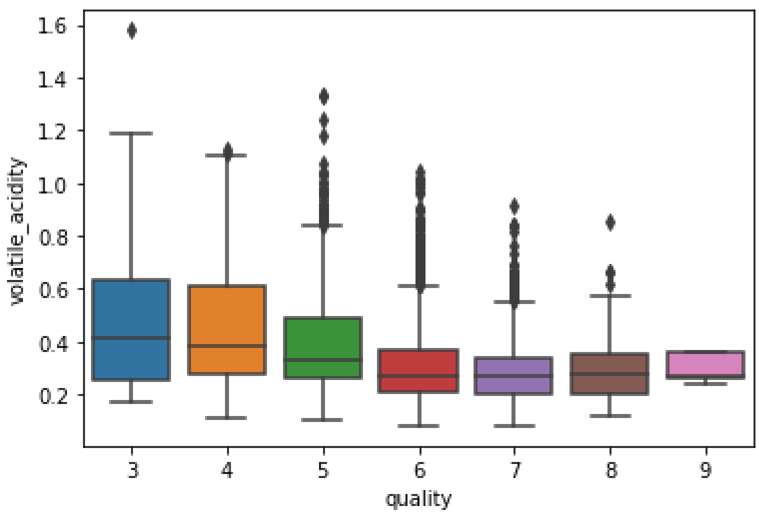

color = red
color = white
total_sulfur_dioxide
0
100
200
300
400
quality
3
4
5
6
7
8
9
color = red
color = white
density
0.99
1.00
1.01
1.02
1.03
1.04
quality
color = red
color = white
pH
2.8
3.0
3.2
3.4
3.6
3.8
4.0
quality

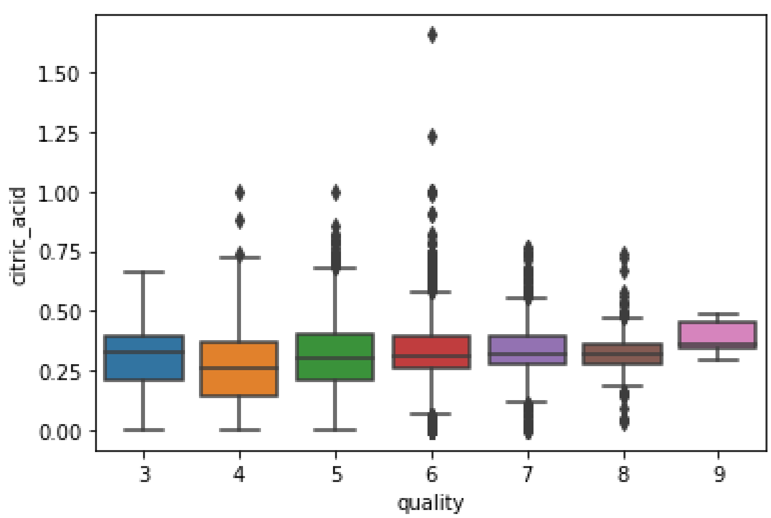

color = red
color = white
sulphates
quality
quality

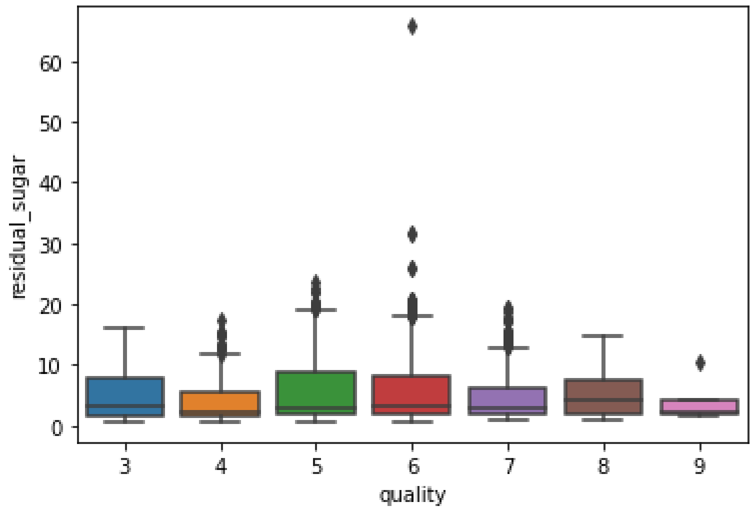

color = red
color = white
alcohol
quality
quality

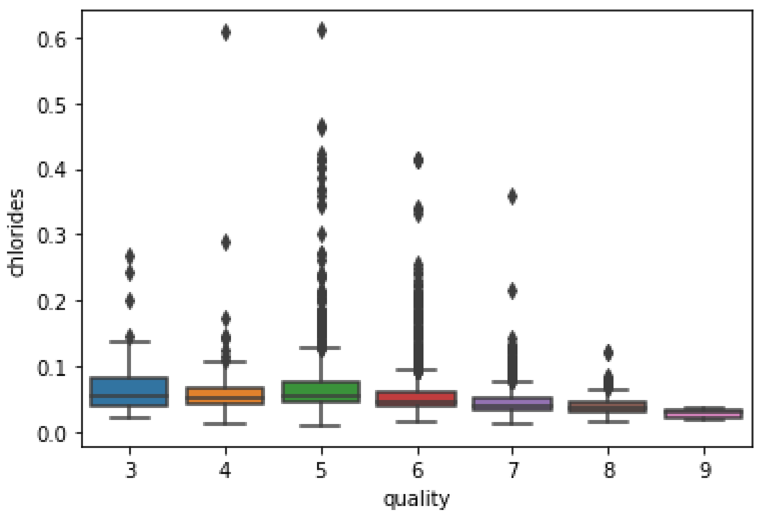
In [55]:
```
for feature in numeric_data:
    sns.boxplot(x='quality',y=feature,data=wdata)
    plt.show()
```

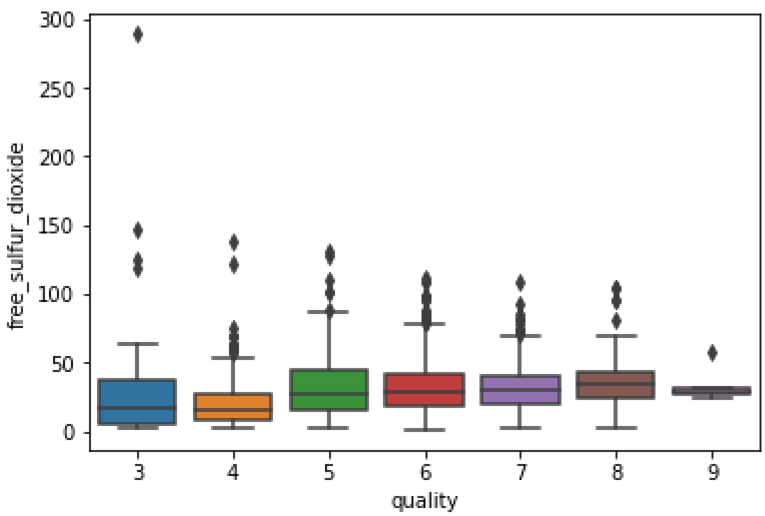

fixed_acidity
16
14
12
10
8
6
4
3
4
5
6
7
8
9
quality

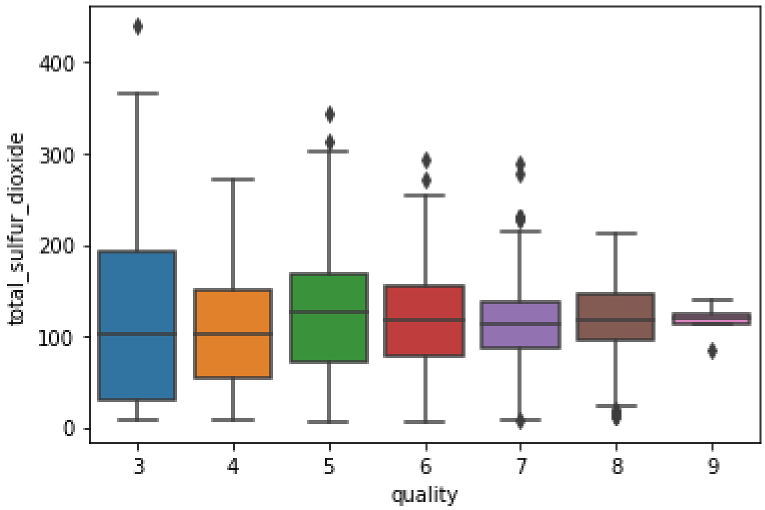

volatile_acidity
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
3
4
5
6
7
8
9
quality

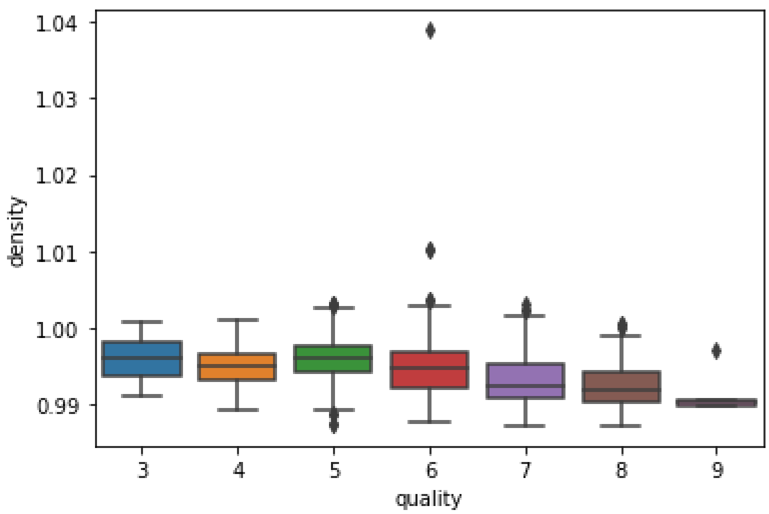

citric_acid
1.50
1.25
1.00
0.75
0.50
0.25
0.00
3
4
5
6
7
8
9
quality

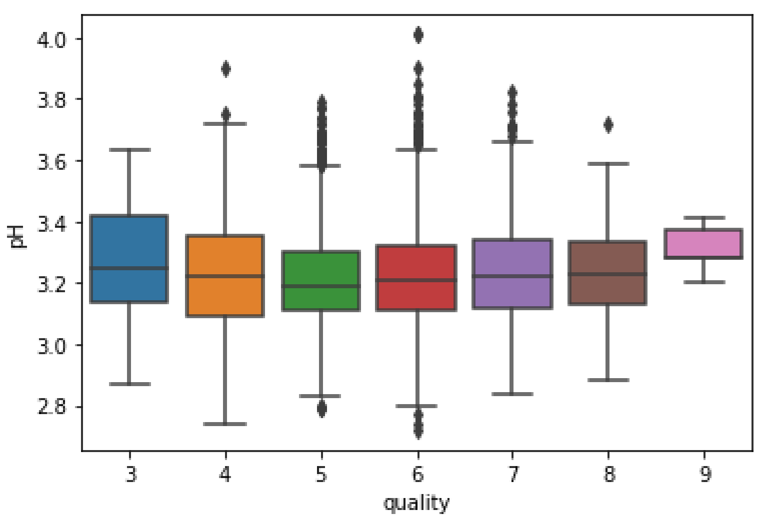

residual_sugar
quality
0
10
20
30
40
50
60
3
4
5
6
7
8
9

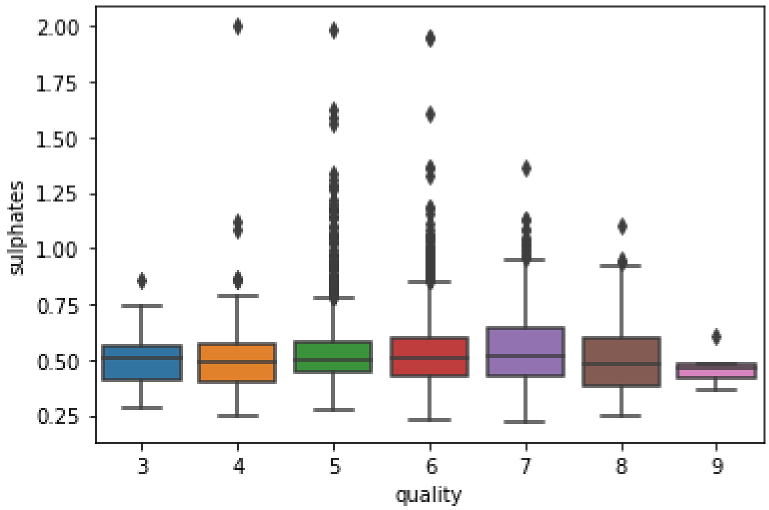

chlorides
quality
0.0
0.1
0.2
0.3
0.4
0.5
0.6
3
4
5
6
7
8
9

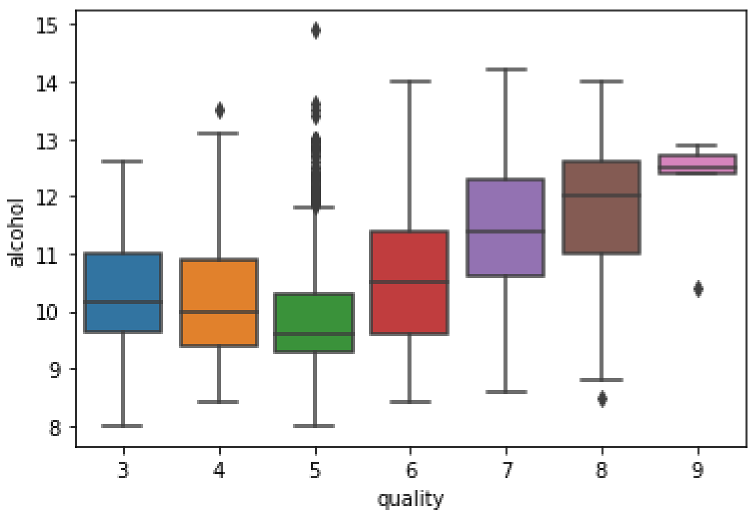

free_sulfur_dioxide
quality
0
50
100
150
200
250
300
3
4
5
6
7
8
9


total_sulfur_dioxide
400
300
200
100
0
3
4
5
6
7
8
9
quality


density
1.04
1.03
1.02
1.01
1.00
0.99
3
4
5
6
7
8
9
quality


pH
4.0
3.8
3.6
3.4
3.2
3.0
2.8
3
4
5
6
7
8
9
quality





In [32]:
```
sns.set_context('talk')
#sns.set_palette(palette)
sns.set_style('white')
```

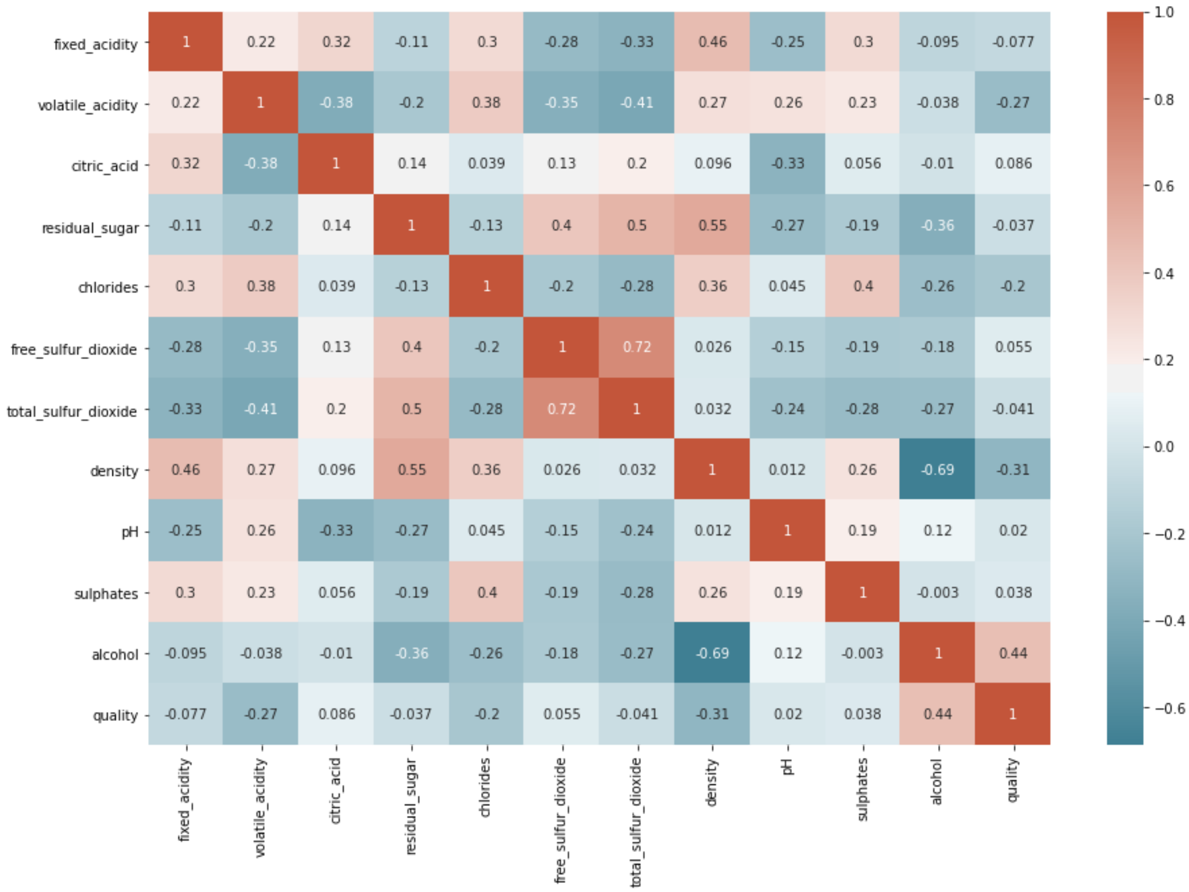
In [34]:
```
sns.pairplot(wdata, hue='color')
```

```
/opt/conda/envs/Python-3.7-OpenCE/lib/python3.7/site-packages/seaborn/distributions.py:369: UserWarning: Default bandwidth for data is 0; skipping density estimation.
  warnings.warn(msg, UserWarning)
/opt/conda/envs/Python-3.7-OpenCE/lib/python3.7/site-packages/seaborn/distributions.py:369: UserWarning: Default bandwidth for data is 0; skipping density estimation.
  warnings.warn(msg, UserWarning)
```

Out[34]:
```
<seaborn.axisgrid.PairGrid at 0x7f0abd314390>
```

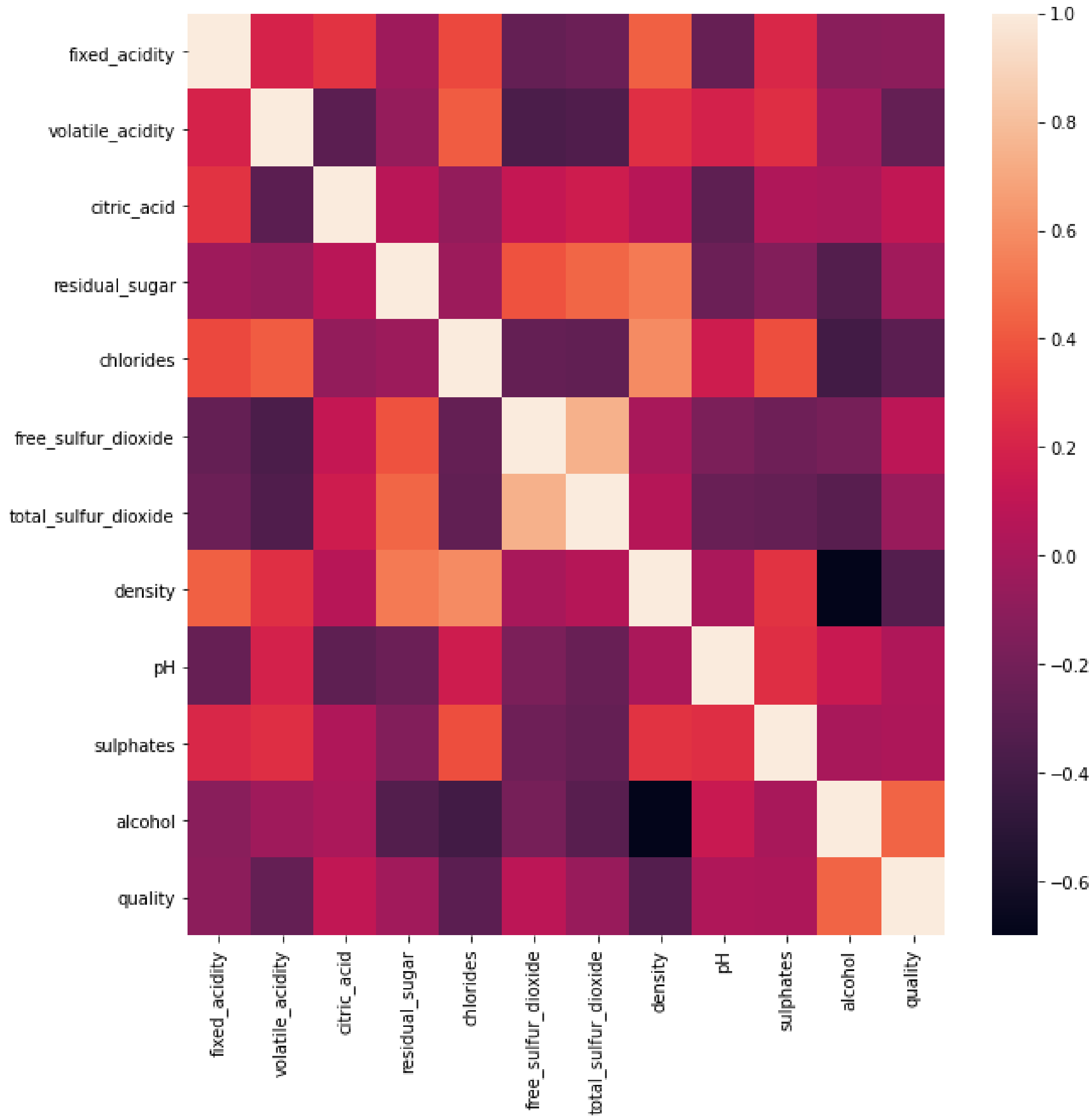
In [11]:
```
# Correlation Matrix
# Will check the correlations between the variables to get a much better under
standing of the relationships between variables
#### there are some variables that are strongly correlated to quality.
### These variables are likely to be most important features in our machine le
arning model

corr = wdata.corr()
plt.subplots(figsize=(15,10))
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns, annot=Tr
ue, cmap=sns.diverging_palette(220, 20, as_cmap=True))
```

Out[11]: <matplotlib.axes._subplots.AxesSubplot at 0x7fad61e697d0>



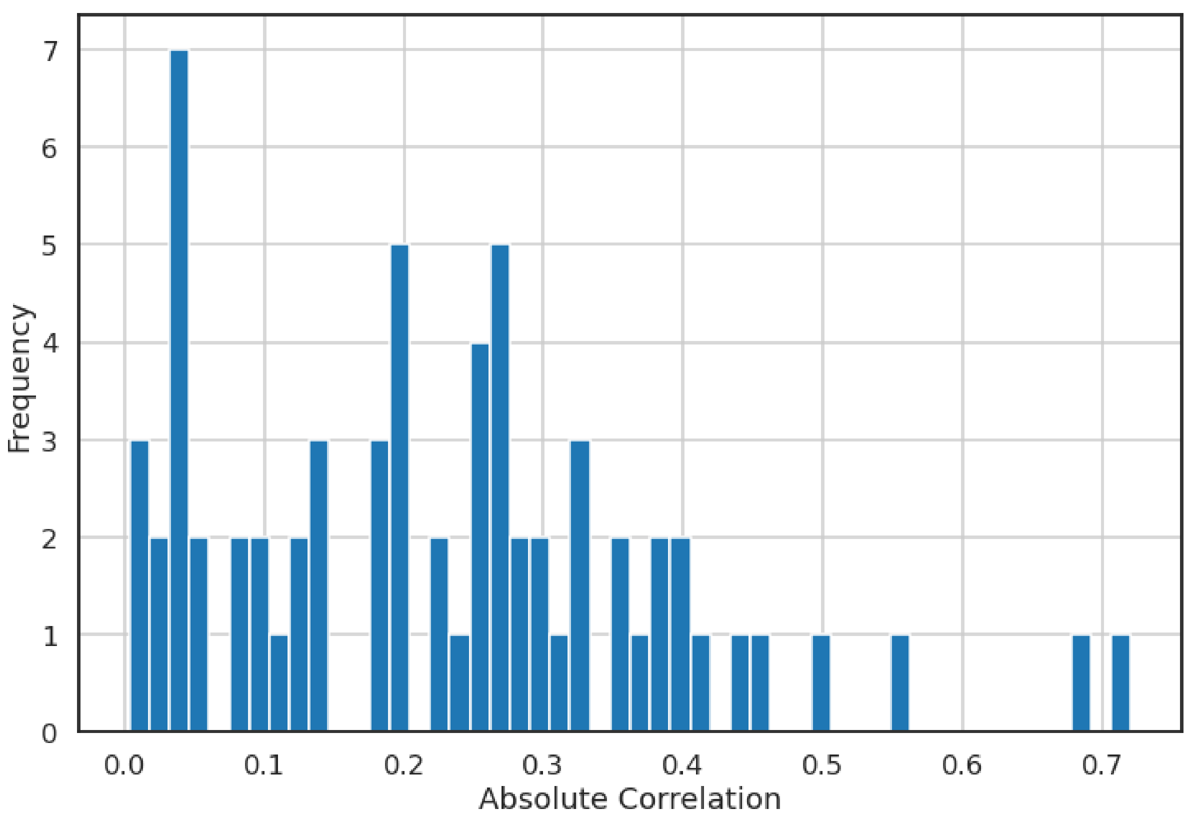
In [12]:
```
# Correlation between features:
plt.figure(figsize=(10,10))
corrmat = wdata.corr(method='spearman')
sns.heatmap(corrmat)
# Findings
# There isn't any strong correlation between features and target variable.
# Alcohol shows moderate correlation.
# Density, Chlorides, volatile acidity show negative correlation which we also
interpreted from inferences made using boxplot
```

Out[12]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fad70ac7390>
```



In [69]:
```
# Calculate the correlation values
feature_cols = wdata.columns[:-1]
corr_values = wdata[feature_cols].corr()

# Simplify by emptying all the data below the diagonal
tril_index = np.tril_indices_from(corr_values)

# Make the unused values NaNs
for coord in zip(*tril_index):
    corr_values.iloc[coord[0], coord[1]] = np.NaN

# Stack the data and convert to a data frame
corr_values = (corr_values
               .stack()
               .to_frame()
               .reset_index()
               .rename(columns={'level_0':'feature1',
                                'level_1':'feature2',
                                0:'correlation'}))

# Get the absolute values for sorting
corr_values['abs_correlation'] = corr_values.correlation.abs()
```

In [70]:
```
sns.set_context('talk')
sns.set_style('white')
ax = corr_values.abs_correlation.hist(bins=50, figsize=(12, 8))
ax.set(xlabel='Absolute Correlation', ylabel='Frequency');
```



In [71]:
```
# The most highly correlated values ->> greater than 0.5
corr_values.sort_values('correlation', ascending=False).query('abs_correlation
>0.5')
### END SOLUTION
```

Out[71]:

	feature1	feature2	correlation	abs_correlation
45	free_sulfur_dioxide	total_sulfur_dioxide	0.720934	0.720934
33	residual_sugar	density	0.552517	0.552517
58	density	alcohol	-0.686745	0.686745

C-2 : DATA CLEANING

In [13]:
```
# Let's check dfor the null values in dataset
print(wdata.isnull().sum().to_string())
### there are no null values ###
```

```
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
color                   0
```

C-3 : FEATURE ENGINEERING

Convert to a Classification Problem

Since the objective is to develop and evalaute various classification models, so I needed to change the output variable to a binary output. For this problem, I defined a bottle of wine as 'good quality' if it had a quality score of 7 or higher, and if it had a score of less than 7, it was deemed 'bad quality'.

Each wine in this dataset is given a “quality” score between 0 and 10. For the purpose of this project, I converted the output to a binary output where each wine is either “good quality” (a score of 7 or higher) or not (a score below 7). The quality of a wine is determined by 11 input variables: These datasets can be viewed as classification or regression tasks. The classes are ordered and not balanced (e.g. there are many more normal wines than excellent or poor ones). Outlier detection algorithms could be used to detect the few excellent or poor wines. Also, we are not sure if all input variables are relevant. So it could be interesting to test feature selection methods.

In [197]:
```
# coverting y (color column) to integer 0,1
# Scikit learn classifiers won't accept a sparse matrix for the prediction col
umn. Thus, either LabelEncoder needs to be used to convert to integers,
# or if DictVectorizer is used, the resulting matrix must be converted to a no
n-sparse array.
# In this case since we had only two labels, we use simple astype(int) code as
below

from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
wdatat=wdata.copy()
wdatat['color'] = le.fit_transform(wdatat.color)
wdatat.sample(5)
```

Out[197]:

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	tot
1614	6.6	0.17	0.38	1.5	0.032	28.0	
808	7.4	0.53	0.12	1.9	0.165	4.0	
3560	9.5	0.21	0.47	1.3	0.039	21.0	
6287	6.7	0.16	0.32	12.5	0.035	18.0	
3247	7.1	0.30	0.49	1.6	0.045	31.0	

In [198]:
```
wdatat['color'].value_counts()
```

Out[198]:
```
1    4898
0    1599
Name: color, dtype: int64
```

In [199]:
```
# Each wine in this dataset is given a “quality” score between 0 and 10.
# For the purpose of this project, I converted the output to a binary output w
here
# Each wine is either “good quality” (a score of 7 or higher) or not (a score
 below 7).
# Create Classification version of target variable
# wdata1 = wdata.copy(deep=True)
wdatat['goodquality'] = [1 if x >= 7 else 0 for x in wdata['quality']]

# DROPPING 'QUALITY' FEATURE AS ITS REPLACED BY 'GOODQUALITY'

wdatat = wdatat.drop('quality', axis=1)
```

In [200]:
```
wdatat.sample(5)
```

Out[200]:

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxide	tot
5467	6.0	0.220	0.25	11.1	0.056	112.0	
2617	7.1	0.180	0.42	1.4	0.045	47.0	
877	7.7	0.715	0.01	2.1	0.064	31.0	
5837	6.4	0.290	0.18	15.0	0.040	21.0	
5384	5.6	0.190	0.31	2.7	0.027	11.0	

In [201]:
```
### check the balace in goodquality
wdatat['goodquality'].value_counts()
```

Out[201]:
```
0    5220
1    1277
Name: goodquality, dtype: int64
```

In [202]:
```
wdatat.describe()
```

Out[202]:

	fixed_acidity	volatile_acidity	citric_acid	residual_sugar	chlorides	free_sulfur_dioxid
count	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.00000
mean	7.215307	0.339666	0.318633	5.443235	0.056034	30.52531
std	1.296434	0.164636	0.145318	4.757804	0.035034	17.74940
min	3.800000	0.080000	0.000000	0.600000	0.009000	1.00000
25%	6.400000	0.230000	0.250000	1.800000	0.038000	17.00000
50%	7.000000	0.290000	0.310000	3.000000	0.047000	29.00000
75%	7.700000	0.400000	0.390000	8.100000	0.065000	41.00000
max	15.900000	1.580000	1.660000	65.800000	0.611000	289.00000

In [203]:
```
# Separate feature variables and target variable
X = wdatat.drop(['goodquality'], axis = 1)
y = wdatat['goodquality']
```

In [171]:
```
### SCALING OPTION1 ###
# Normalize feature variables
from sklearn.preprocessing import StandardScaler
#X_features = X
#X = StandardScaler().fit_transform(X)
```

In [204]:
```
### SCALING OPTION2 ###
from sklearn.preprocessing import MinMaxScaler
scalar = MinMaxScaler()
X = scalar.fit_transform(X)
```

In [205]:
```
pd.DataFrame(X).describe()
```

Out[205]:

	0	1	2	3	4	5	
count	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000000	6497.000C
mean	0.282257	0.173111	0.191948	0.074283	0.078129	0.102518	0.2528
std	0.107143	0.109758	0.087541	0.072972	0.058195	0.061630	0.1302
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000C
25%	0.214876	0.100000	0.150602	0.018405	0.048173	0.055556	0.1635
50%	0.264463	0.140000	0.186747	0.036810	0.063123	0.097222	0.258C
75%	0.322314	0.213333	0.234940	0.115031	0.093023	0.138889	0.3456
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000C

SPLITTING THE DATA

we will Split the data into train and test data sets. This can be done using any method, but consider using Scikit-learn's StratifiedShuffleSplit to maintain the same ratio of predictor classes. Regardless of methods used to split the data, compare the ratio of classes in both the train and test splits.

In [173]:
```
### OPTION 1 SPLITTING - Normal Split ###
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.3, random
_state=50)
```

In [222]:
```
### OPTION 2 SPLITTING - StratifiedShuffleSplit ###
from sklearn.model_selection import StratifiedShuffleSplit
feature_cols = [x for x in wdatat.columns if x not in 'goodquality']
# Split the data into two parts with 1000 points in the test data
# This creates a generator
strat_shuff_split = StratifiedShuffleSplit(n_splits=1, test_size=0.3, random_s
tate=50)

# Get the index values from the generator

train_idx, test_idx = next(strat_shuff_split.split(wdatat[feature_cols], wdata
t['goodquality']))

# Create the data sets
X_train = wdatat.loc[train_idx, feature_cols]
y_train = wdatat.loc[train_idx, 'goodquality']

X_test = wdatat.loc[test_idx, feature_cols]
y_test = wdatat.loc[test_idx, 'goodquality']
```

In [223]:
```
wdatat['goodquality'].value_counts(normalize=True)
```

Out[223]:
```
0    0.803448
1    0.196552
Name: goodquality, dtype: float64
```

In [225]:
```
y_train.value_counts(normalize=True)
```

Out[225]:
```
0    0.803387
1    0.196613
Name: goodquality, dtype: float64
```

In [224]:
```
y_test.value_counts(normalize=True)
```

Out[224]:
```
0    0.80359
1    0.19641
Name: goodquality, dtype: float64
```

D) Summary of training at least three different classifier models, preferably of different nature in explainability and predictability. For example, you can start with a simple logistic regression as a baseline, adding other models or ensemble models. Preferably, all your models use the same training and test splits, or the same cross-validation me

We will now fit a logistic regression model without any regularization using all of the features. We will also use cross validation to determine the hyperparameters, fit models using L1, and L2 regularization and compare/evalue each of these models as well.

MODEL 1 >>> Standard Logistic Regression, L1 Regularized & L2 Regularized

MODEL 2 >>> Support Vector Machine (SVM)

MODEL 3 >>> Decision Tree & GridSearchCV

MODEL 1 >>> Standard Logistic Regression, L1 Regularized & L2 Regularized

In [226]:
```
# Standard logistic regression
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, roc_auc_score
lr = LogisticRegression(solver='liblinear').fit(X_train, y_train)
y_test_p_lr=lr.predict(X_test)
y_train_p_lr=lr.predict(X_train)
lr.coef_
```

Out[226]:
```
array([[-0.02308799, -3.38401235, -0.03411285,  0.04823298, -1.6922069 ,
         0.01330668, -0.00536188, -4.04852663, -0.65129519,  1.30774428,
         0.87766094, -0.19464785]])
```

In [105]:
```
confusion_matrix(y_test, y_test_p_lr)
```

Out[105]:
```
array([[1477,   62],
       [ 322,   89]])
```

In [106]:
```
accuracy_score(y_test, y_test_p_lr)
```

Out[106]:
```
0.803076923076923
```

In [107]:
```
# The error on the training and test data sets
train_test_lr_error = pd.concat([measure_error(y_train, y_train_p_lr, 'train'
),
                               measure_error(y_test, y_test_p_lr, 'test')],
                               axis=1)

train_test_lr_error
```

Out[107]:

	train	test
accuracy	0.821861	0.803077
precision	0.594595	0.589404
recall	0.203233	0.216545
f1	0.302926	0.316726

In [108]:
```
print(classification_report(y_train, y_train_p_lr))
```

```
              precision    recall  f1-score   support

           0       0.84      0.97      0.90      3681
           1       0.59      0.20      0.30       866

    accuracy                           0.82      4547
   macro avg       0.72      0.59      0.60      4547
weighted avg       0.79      0.82      0.78      4547
```

In [227]:
```
print(classification_report(y_test, y_test_p_lr))
```

```
              precision    recall  f1-score   support

           0       0.84      0.96      0.89      1567
           1       0.56      0.23      0.33       383

    accuracy                           0.81      1950
   macro avg       0.70      0.59      0.61      1950
weighted avg       0.78      0.81      0.78      1950
```

In [228]:
```
# L1 regularized logistic regression
from sklearn.linear_model import LogisticRegressionCV
lrl1 = LogisticRegressionCV(Cs=10, cv=4, penalty='l1', solver='liblinear').fit
(X_train, y_train)
y_test_p_lrl1=lrl1.predict(X_test)
y_train_p_lrl1=lrl1.predict(X_train)
lrl1.coef_
```

Out[228]:
```
array([[ 1.49435344e-01, -3.67815311e+00, -2.08638263e-01,
         6.69010295e-02, -9.97108097e+00,  1.39583730e-02,
        -5.15169951e-03, -7.24327084e+00,  7.60464520e-01,
         1.69965980e+00,  9.37068926e-01, -7.52412210e-02]])
```

In [111]:
```
confusion_matrix(y_test, y_test_p_lrl1)
```

Out[111]:
```
array([[1475,   64],
       [ 319,   92]])
```

In [112]:
```
accuracy_score(y_test, y_test_p_lrl1)
```

Out[112]:
```
0.8035897435897436
```

In [94]:
```
# The error on the training and test data sets
train_test_lrl1_error = pd.concat([measure_error(y_train, y_train_p_lrl1, 'tra
in'),
                              measure_error(y_test, y_test_p_lrl1, 'test')],
                              axis=1)

train_test_lrl1_error
```

Out[94]:

	train	test
accuracy	0.822740	0.808205
precision	0.578534	0.603352
recall	0.255196	0.262774
f1	0.354167	0.366102

In [116]:
```
print(classification_report(y_train, y_train_p_lrl1))
```

```
              precision    recall  f1-score   support

           0       0.84      0.96      0.90      3681
           1       0.58      0.21      0.31       866

    accuracy                           0.82      4547
   macro avg       0.71      0.59      0.60      4547
weighted avg       0.79      0.82      0.78      4547
```

In [229]:
```
print(classification_report(y_test, y_test_p_lrl1))
```

```
              precision    recall  f1-score   support

           0       0.84      0.95      0.89      1567
           1       0.55      0.27      0.36       383

    accuracy                           0.81      1950
   macro avg       0.70      0.61      0.63      1950
weighted avg       0.78      0.81      0.79      1950
```

In [230]:
```
# L2 regularized logistic regression
from sklearn.linear_model import LogisticRegressionCV
lrl2 = LogisticRegressionCV(Cs=10, cv=4, penalty='l2', solver='liblinear').fit
(X_train, y_train)
y_test_p_lrl2=lrl2.predict(X_test)
y_train_p_lrl2=lrl2.predict(X_train)
lrl2.coef_
```

Out[230]:
```
array([[ 1.61722515e-01, -3.70870366e+00, -2.43352240e-01,
         6.84485714e-02, -8.41867258e+00,  1.40428122e-02,
        -5.21191047e-03, -7.90292366e+00,  8.98749353e-01,
         1.69727696e+00,  9.49093785e-01,  1.28591709e-02]])
```

In [81]:
```
# The error on the training and test data sets

train_test_lrl2_error = pd.concat([measure_error(y_train, y_train_p_lrl2, 'tra
in'),
                                  measure_error(y_test, y_test_p_lrl2, 'test')],
                                  axis=1)

train_test_lrl2_error
```

Out[81]:

	train	test
accuracy	0.822520	0.809231
precision	0.576227	0.610169
recall	0.257506	0.262774
f1	0.355946	0.367347

In [34]:
```
confusion_matrix(y_test, y_test_p_lrl2)
```

Out[34]:
```
array([[1470,   69],
       [ 303,  108]])
```

In [35]:
```
accuracy_score(y_test, y_test_p_lrl2)
```

Out[35]:
```
0.8092307692307692
```

In [89]:
```
print(classification_report(y_train, y_train_p_lrl2))
```

```
              precision    recall  f1-score   support

           0       0.85      0.96      0.90      3681
           1       0.58      0.26      0.36       866

    accuracy                           0.82      4547
   macro avg       0.71      0.61      0.63      4547
weighted avg       0.79      0.82      0.79      4547
```

In [231]:
```
print(classification_report(y_test, y_test_p_lrl2))
```

```
              precision    recall  f1-score   support

           0       0.84      0.95      0.89      1567
           1       0.55      0.27      0.36       383

    accuracy                           0.81      1950
   macro avg       0.69      0.61      0.62      1950
weighted avg       0.78      0.81      0.79      1950
```

For each model, calculate the following error metrics: Accuracy Precision Recall F-score Confusion Matrix Decide how to combine the multi-class metrics into a single value for each model.

MODEL 2 >>> Support Vector Machine (SVM)

In [ ]:

In [232]:
```
from sklearn.svm import LinearSVC
lscv = LinearSVC()
lscv.fit(X_train, y_train)
```

```
/opt/conda/envs/Python-3.7-OpenCE/lib/python3.7/site-packages/sklearn/svm/_ba
se.py:977: ConvergenceWarning: Liblinear failed to converge, increase the num
ber of iterations.
  "the number of iterations.", ConvergenceWarning)
```

Out[232]:
```
LinearSVC()
```

In [119]:
```
lscv.coef_
```

Out[119]:
```
array([[ 0.01042635, -1.240539  , -0.27502932,  0.02136014, -0.62806316,
         0.01712583, -0.00208026, -1.38674335, -0.31036954,  0.88079438,
         0.29087225, -0.1104003 ]])
```

In [235]:
```
y_test_p_LSCV = lscv.predict(X_test)
y_train_p_LSCV = lscv.predict(X_train)
confusion_matrix(y_test, y_test_p_LSCV)
```

Out[235]:
```
array([[1560,    7],
       [ 374,    9]])
```

In [121]:
```
### The decision tree predicts a little better on the training data than the t
est data, which is consistent with (mild) overfitting. Also notice the perfect
recall score for the training data. In many instances, this prediction differe
nce is even greater than that seen here.
# The error on the training and test data sets
train_test_lscv_error = pd.concat([measure_error(y_train, y_train_p_LSCV, 'tra
in'),
                              measure_error(y_test, y_test_p_LSCV, 'test')],
                              axis=1)

train_test_lscv_error
```

Out[121]:

	train	test
accuracy	0.768199	0.778462
precision	0.406931	0.475177
recall	0.474596	0.489051
f1	0.438166	0.482014

In [122]:
```
accuracy_score(y_test, y_test_p_LSCV)
```

Out[122]: 0.7784615384615384

In [149]:
```
print(classification_report(y_train, y_train_p_LSCV))
```

```
              precision    recall  f1-score   support

           0       0.87      0.84      0.85      3681
           1       0.41      0.47      0.44       866

    accuracy                           0.77      4547
   macro avg       0.64      0.66      0.65      4547
weighted avg       0.78      0.77      0.77      4547
```

In [236]:
```
print(classification_report(y_test, y_test_p_LSCV))
```

```
              precision    recall  f1-score   support

           0       0.81      1.00      0.89      1567
           1       0.56      0.02      0.05       383

    accuracy                           0.80      1950
   macro avg       0.68      0.51      0.47      1950
weighted avg       0.76      0.80      0.72      1950
```

MODEL 3 >>> Decision Tree & GridSearchCV

1- Decision Tree - DecisionTreeClassifier

In [45]:
```
from sklearn.metrics import classification_report
```

In [237]:
```
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier(random_state=1)
dtc.fit(X_train, y_train)
dtc.tree_.node_count, dtc.tree_.max_depth
```

Out[237]: (1059, 25)

In [152]:
```
### The decision tree predicts a little better on the training data than the t
est data, which is consistent with (mild) overfitting. Also notice the perfect
recall score for the training data. In many instances, this prediction differe
nce is even greater than that seen here.
# The error on the training and test data sets
train_test_dtc_error = pd.concat([measure_error(y_train, y_train_p_dtc, 'trai
n'),
                              measure_error(y_test, y_test_p_dtc, 'test')],
                              axis=1)

train_test_dtc_error
```

Out[152]:

	train	test
accuracy	1.0	0.826154
precision	1.0	0.588670
recall	1.0	0.581509
f1	1.0	0.585067

In [239]:
```
y_test_p_dtc = dtc.predict(X_test)
y_train_p_dtc = dtc.predict(X_train)
print(classification_report(y_test, y_test_p_dtc))
```

```
              precision    recall  f1-score   support

           0       0.91      0.90      0.90      1567
           1       0.60      0.62      0.61       383

    accuracy                           0.84      1950
   macro avg       0.75      0.76      0.76      1950
weighted avg       0.85      0.84      0.84      1950
```

In [155]:
```
print(classification_report(y_train, y_train_p_dtc))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3681
           1       1.00      1.00      1.00       866

    accuracy                           1.00      4547
   macro avg       1.00      1.00      1.00      4547
weighted avg       1.00      1.00      1.00      4547
```

In [156]:
```
confusion_matrix(y_test, y_test_p_dtc)
```

Out[156]:
```
array([[1372,  167],
       [ 172,  239]])
```

In [131]:
```
accuracy_score(y_test, y_test_p_dtc)
```

Out[131]:
```
0.8261538461538461
```

2- Decision Tree - Random Forest

In [240]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=1)
rfc.fit(X_train, y_train)
```

Out[240]:
```
RandomForestClassifier(random_state=1)
```

In [133]:
```
### The decision tree predicts a little better on the training data than the t
est data, which is consistent with (mild) overfitting. Also notice the perfect
recall score for the training data. In many instances, this prediction differe
nce is even greater than that seen here.
# The error on the training and test data sets
train_test_rfc_error = pd.concat([measure_error(y_train, y_train_p_rfc, 'trai
n'),
                              measure_error(y_test, y_test_p_rfc, 'test')],
                              axis=1)

train_test_rfc_error
```

Out[133]:

	train	test
accuracy	1.0	0.867179
precision	1.0	0.781481
recall	1.0	0.513382
f1	1.0	0.619677

In [134]:
```
confusion_matrix(y_test, y_test_p_rfc)
```

Out[134]:
```
array([[1480,   59],
       [ 200,  211]])
```

In [137]:
```
accuracy_score(y_test, y_test_p_rfc)
```

Out[137]:
```
0.8676923076923077
```

In [241]:
```
y_test_p_rfc = rfc.predict(X_test)
print(classification_report(y_test, y_test_p_rfc))
```

```
              precision    recall  f1-score   support

           0       0.90      0.97      0.93      1567
           1       0.80      0.56      0.66       383

    accuracy                           0.89      1950
   macro avg       0.85      0.76      0.80      1950
weighted avg       0.88      0.89      0.88      1950
```

In [139]:
```
y_train_p_rfc = rfc.predict(X_train)
print(classification_report(y_train, y_train_p_rfc))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3681
           1       1.00      1.00      1.00       866

    accuracy                           1.00      4547
   macro avg       1.00      1.00      1.00      4547
weighted avg       1.00      1.00      1.00      4547
```

3- Decision Tree - GridSearchCV

Using grid search with cross validation, find a decision tree that performs well on the test data set.

In [242]:
```
from sklearn.model_selection import GridSearchCV

param_grid = {'max_depth':range(1, dtc.tree_.max_depth+1, 2),
              'max_features': range(1, len(dtc.feature_importances_)+1)}

gscv = GridSearchCV(DecisionTreeClassifier(random_state=42),
                    param_grid=param_grid,
                    scoring='accuracy',
                    n_jobs=-1)

gscv= gscv.fit(X_train, y_train)
y_train_p_gscv = gscv.predict(X_train)
y_test_p_gscv = gscv.predict(X_test)
```

In [192]:
```
gscv.best_estimator_.tree_.node_count, gscv.best_estimator_.tree_.max_depth
```

Out[192]: (893, 13)

In [142]:
```
### TRAIN VS TEST ERROR - A function to return error metrics
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
score

def measure_error(y_true, y_pred, label):
    return pd.Series({'accuracy':accuracy_score(y_true, y_pred),
                      'precision': precision_score(y_true, y_pred),
                      'recall': recall_score(y_true, y_pred),
                      'f1': f1_score(y_true, y_pred)},
                      name=label)
```

In [143]:
```
### The decision tree predicts a little better on the training data than the t
est data, which is consistent with (mild) overfitting. Also notice the perfect
recall score for the training data. In many instances, this prediction differe
nce is even greater than that seen here.
# The error on the training and test data sets
train_test_gscv_error = pd.concat([measure_error(y_train, y_train_p_gscv, 'tra
in'),
                              measure_error(y_test, y_test_p_gscv, 'test')],
                              axis=1)

train_test_gscv_error
```

Out[143]:

	train	test
accuracy	0.981526	0.830256
precision	0.942308	0.597087
recall	0.961894	0.598540
f1	0.952000	0.597813

In [144]:
```
confusion_matrix(y_test, y_test_p_gscv)
```

Out[144]:
```
array([[1373,  166],
       [ 165,  246]])
```

In [145]:
```
accuracy_score(y_test, y_test_p_gscv)
```

Out[145]:
```
0.8302564102564103
```

In [146]:
```
print(classification_report(y_train, y_train_p_gscv))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      3681
           1       0.94      0.96      0.95       866

    accuracy                           0.98      4547
   macro avg       0.97      0.97      0.97      4547
weighted avg       0.98      0.98      0.98      4547
```

In [243]:
```
print(classification_report(y_test, y_test_p_gscv))
```

```
              precision    recall  f1-score   support

           0       0.91      0.90      0.90      1567
           1       0.60      0.63      0.61       383

    accuracy                           0.85      1950
   macro avg       0.75      0.76      0.76      1950
weighted avg       0.85      0.85      0.85      1950
```

E) A paragraph explaining which of your classifier models you recommend as a final model that best fits your needs in terms of accuracy and explainability.

The objective of the analysis was to predict the quality of wine. The accuracy of the model has to be very high. At the same time since the intent is to reject low quality wines and select good quality wines , its important to have good recall for good quality wines. I will compare my models based on these two metrices (overall accuracy and recall(red)) and select the final model.

Here is summary of metrices for various models

Model Accuracy Recall(1)

Logistic Regression (3 variations) lr 80 22

lrl1 80 22

lrl2 81 26

Support Vector Macine svm 78 49

Decision Tree (3 variations)

dct 83 58

rfc 87 52

gscv 83 60

Overall Decision tree is giving best performance. Further using splitshuffle splitting we further enhance its performance

Model Accuracy Recall(1)

dct 84 62

rfc 89 56

gscv 85 63

Finally takeing into account both metrices , I finalize to use RANDOM FOREST for my prediction.

F) Summary Key Findings and Insights, which walks your reader through the main drivers of your model and insights from your data derived from your classifier model.¶

The objective of the analysis was to predict the quality of wine. The accuracy of the model has to be very high. At the same time since the intent is to reject low quality wines and select good quality wines , its important to have good recall for good quality wines. I evaluated my models based on these two metrices (overall accuracy and recall(red)) and select the final model.

While the accuracy of the model is quite good , recall is very poor specifically in Logistic regression. If the objective of the model is very much dependent on recall, its worthwhile to explore why it is so low. The best recall value i could get in 63% which is not very satifactory.

G) Suggestions for next steps in analyzing this data, which may include suggesting revisiting this model after adding specific data features that may help you achieve a better explanation or a better prediction.

There are other models which could have been tried but for the purpose of this assignment I just tried these three models

To futher enhance accuracy/recall we could have used multiple bagging and boosting techniques

I spent little time on feature engineering and applied only few techniques. Time permitting , i could have done more feature engineering like elimination of few features which are not relavent based on data exporation I have done.

Overall despite achieving good overall accuracy (89%) there are other tools & techniques which could have further improved my model.

In [ ]:

```
```