Dataset Information

The two datasets are related to red and white variants of the Portuguese "Vinho Verde" wine. These datasets can be viewed as classification or regression tasks. The classes are ordered and not balanced (e.g. there are munch 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. Two datasets were combined and few values were randomly removed.

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) \

Import modules

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
%matplotlib inline
warnings.filterwarnings('ignore')
```

Loading the dataset

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

Out[2]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	white	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	white	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	white	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

In [3]:
```
# statistical info
df.describe()
```

Out[3]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
count	6487.000000	6489.000000	6494.000000	6495.000000	6495.000000	6497.000000	6497.000000	6497.000000	6488.000000	6493.000000	6497.000000	6497.000000
mean	7.216579	0.339691	0.318722	5.444326	0.056042	30.525319	115.744574	0.994697	3.218395	0.531215	10.491801	5.818378
std	1.296750	0.164649	0.145265	4.758125	0.035036	17.749400	56.521855	0.002999	0.160748	0.148814	1.192712	0.873255
min	3.800000	0.080000	0.000000	0.600000	0.009000	1.000000	6.000000	0.987110	2.720000	0.220000	8.000000	3.000000
25%	6.400000	0.230000	0.250000	1.800000	0.038000	17.000000	77.000000	0.992340	3.110000	0.430000	9.500000	5.000000
50%	7.000000	0.290000	0.310000	3.000000	0.047000	29.000000	118.000000	0.994890	3.210000	0.510000	10.300000	6.000000
75%	7.700000	0.400000	0.390000	8.100000	0.065000	41.000000	156.000000	0.996990	3.320000	0.600000	11.300000	6.000000
max	15.900000	1.580000	1.660000	65.800000	0.611000	289.000000	440.000000	1.038980	4.010000	2.000000	14.900000	9.000000

In [4]:
```
# datatype info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   type                  6497 non-null   object 
 1   fixed acidity         6487 non-null   float64
 2   volatile acidity      6489 non-null   float64
 3   citric acid           6494 non-null   float64
 4   residual sugar        6495 non-null   float64
 5   chlorides             6495 non-null   float64
 6   free sulfur dioxide   6497 non-null   float64
 7   total sulfur dioxide  6497 non-null   float64
 8   density               6497 non-null   float64
 9   pH                    6488 non-null   float64
 10  sulphates             6493 non-null   float64
 11  alcohol               6497 non-null   float64
 12  quality               6497 non-null   int64  
dtypes: float64(11), int64(1), object(1)
memory usage: 660.0+ KB
```

Preprocessing the dataset

In [5]:
```
# check for null values
df.isnull().sum()
```

Out[5]:
```
type                     0
fixed acidity           10
volatile acidity         8
citric acid              3
residual sugar           2
chlorides                2
free sulfur dioxide      0
total sulfur dioxide     0
density                  0
pH                       9
sulphates                4
alcohol                  0
quality                  0
dtype: int64
```

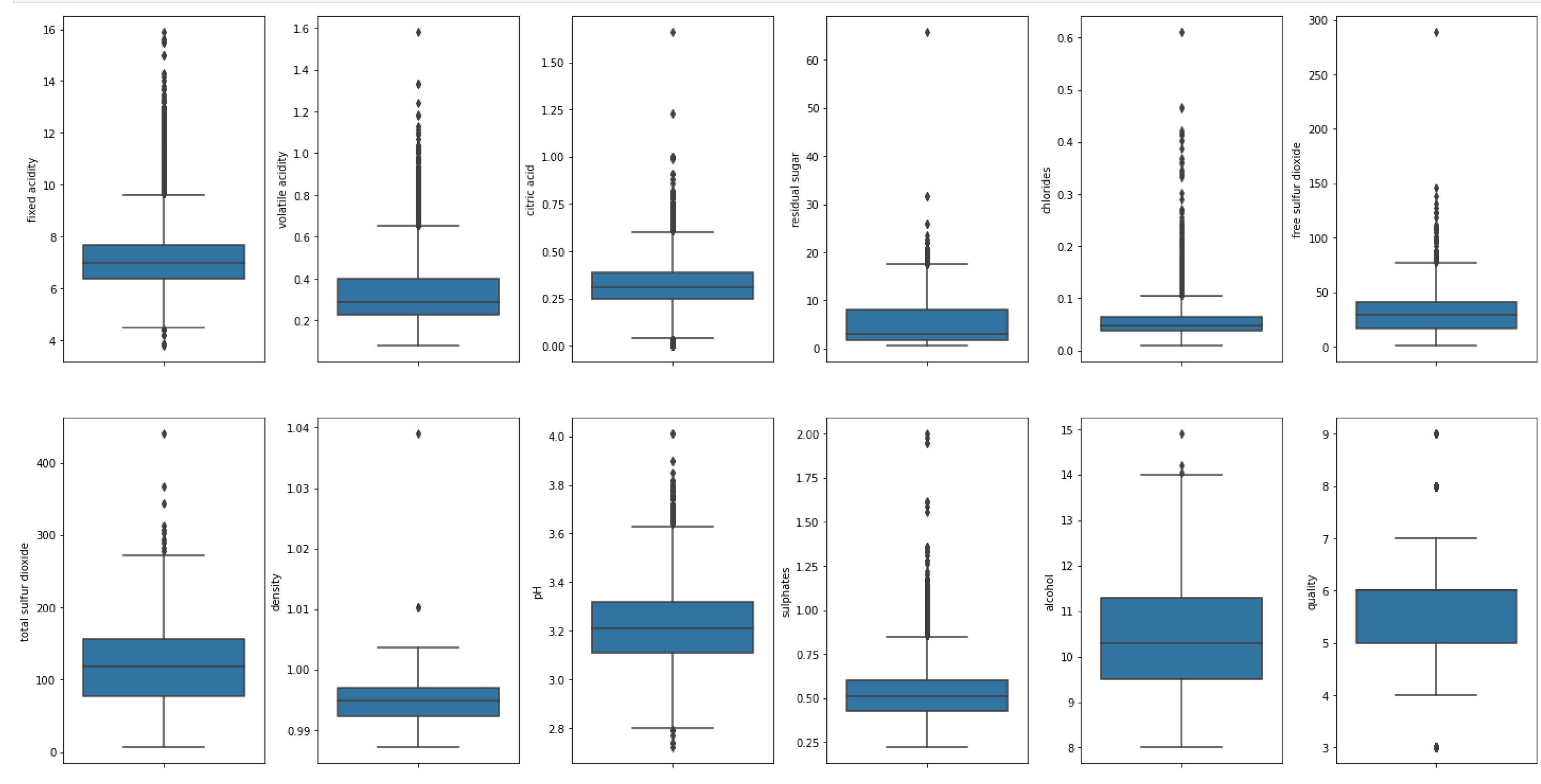
In [6]:
```
# fill the missing values
for col, value in df.items():
    if col != 'type':
        df[col] = df[col].fillna(df[col].mean())
```

In [7]:
```
df.isnull().sum()
```

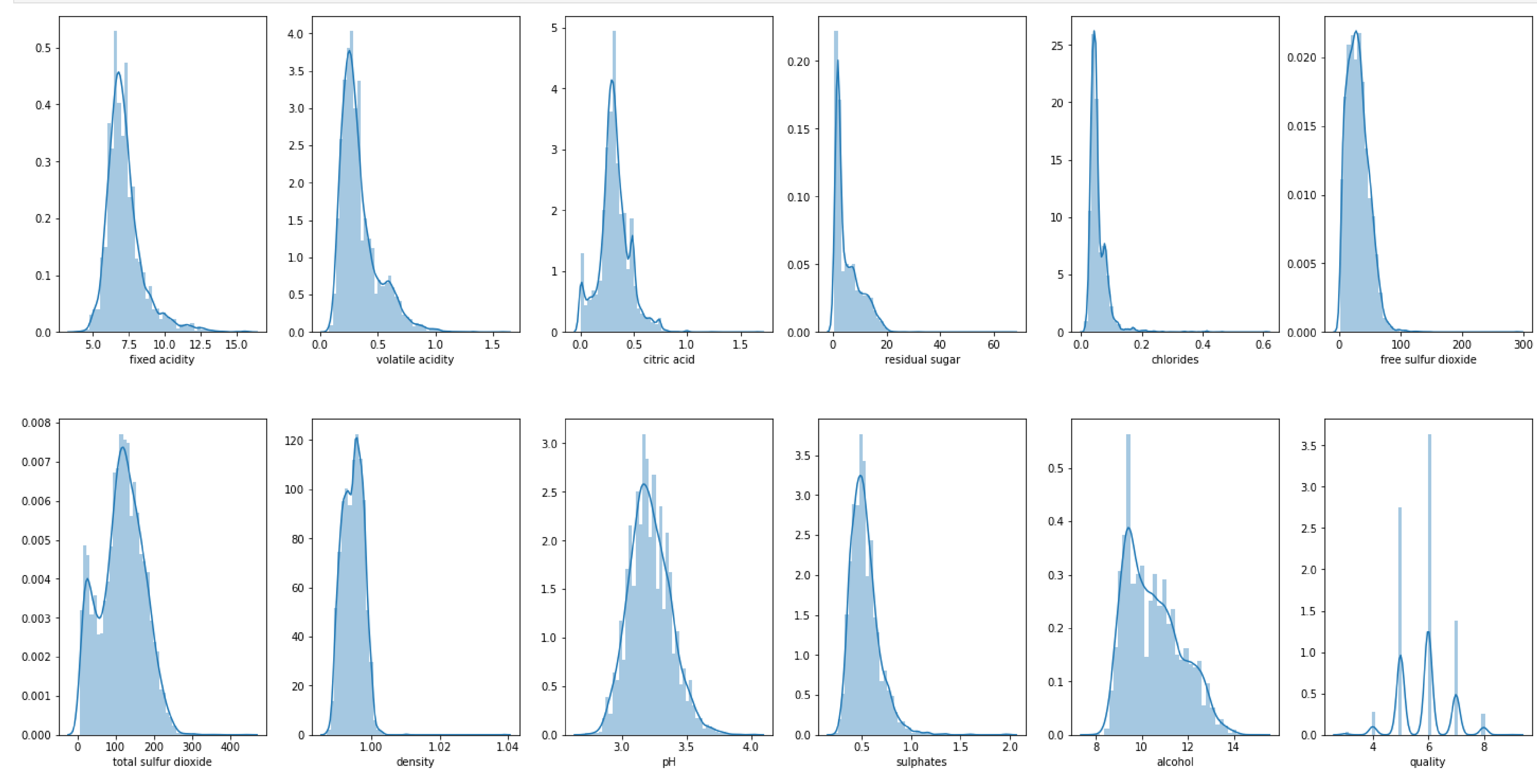
Out[7]:
```
type                    0
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

Exploratory Data Analysis

In [8]:
```
# create box plots
fig, ax = plt.subplots(ncols=6, nrows=2, figsize=(20,10))
index = 0
ax = ax.flatten()

for col, value in df.items():
    if col != 'type':
        sns.boxplot(y=col, data=df, ax=ax[index])
        index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```



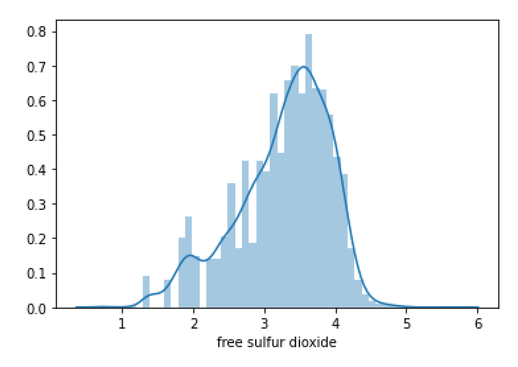
In [9]:
```
# create dist plot
fig, ax = plt.subplots(ncols=6, nrows=2, figsize=(20,10))
index = 0
ax = ax.flatten()

for col, value in df.items():
    if col != 'type':
        sns.distplot(value, ax=ax[index])
        index += 1
plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```



In [10]:
```
# log transformation
df['free sulfur dioxide'] = np.log(1 + df['free sulfur dioxide'])
```

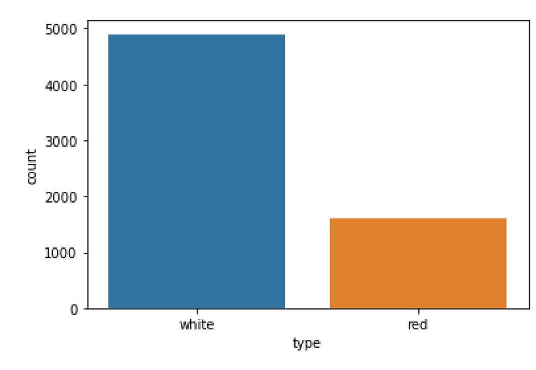
In [11]:
```
sns.distplot(df['free sulfur dioxide'])
```

Out[11]: <AxesSubplot:xlabel='free sulfur dioxide'>



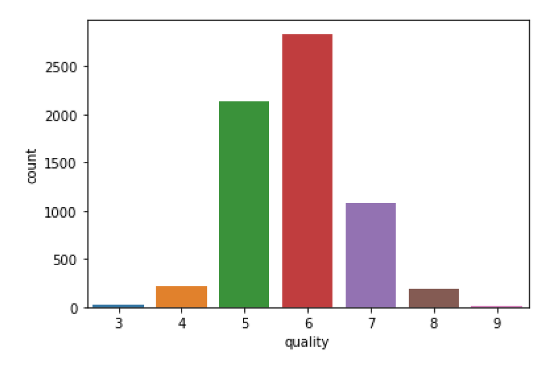
In [12]:
```
sns.countplot(df['type'])
```

Out[12]: <AxesSubplot:xlabel='type', ylabel='count'>



In [13]:
```
sns.countplot(df['quality'])
```

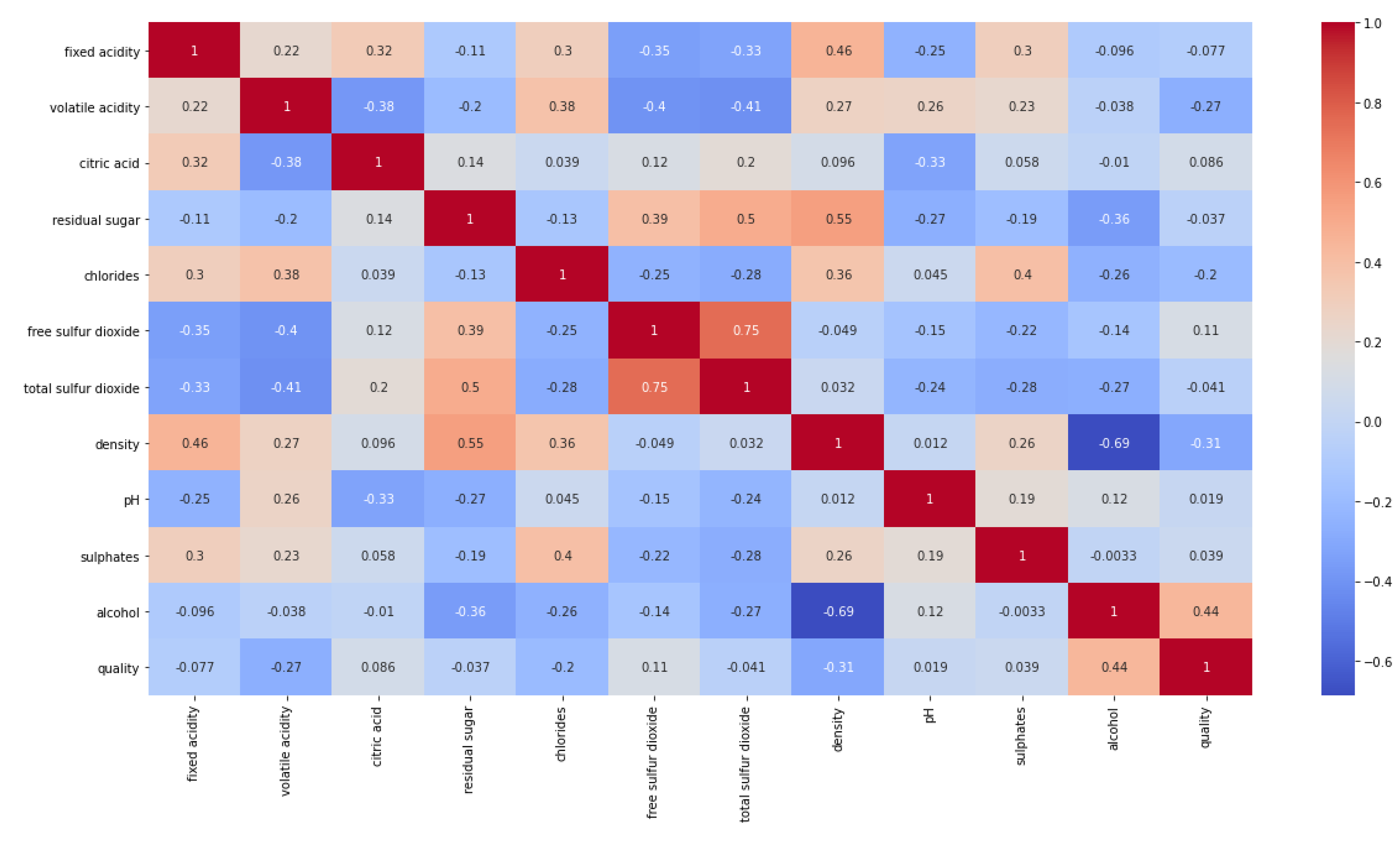
Out[13]: <AxesSubplot:xlabel='quality', ylabel='count'>



Coorelation Matrix

In [14]:
```
corr = df.corr()
plt.figure(figsize=(20,10))
sns.heatmap(corr, annot=True, cmap='coolwarm')
```

Out[14]: <AxesSubplot:>



Input Split

In [15]:
```
X = df.drop(columns=['type', 'quality'])
y = df['quality']
```

Class Imbalancement

In [16]:
```
y.value_counts()
```

Out[16]:
```
6    2836
5    2138
7    1079
4     216
8     193
3      30
9       5
Name: quality, dtype: int64
```

In [17]:
```
from imblearn.over_sampling import SMOTE
oversample = SMOTE(k_neighbors=4)
# transform the dataset
X, y = oversample.fit_resample(X, y)
```

In [18]:
```
y.value_counts()
```

Out[18]:
```
9    2836
8    2836
7    2836
6    2836
5    2836
4    2836
3    2836
Name: quality, dtype: int64
```

Model Training

In [19]:
```
# classify function
from sklearn.model_selection import cross_val_score, train_test_split
def classify(model, X, y):
    x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
    # train the model
    model.fit(x_train, y_train)
    print("Accuracy:", model.score(x_test, y_test) * 100)

    # cross-validation
    score = cross_val_score(model, X, y, cv=5)
    print("CV Score:", np.mean(score)*100)
```

In [20]:
```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
classify(model, X, y)
```

```
Accuracy: 33.568406205923836
CV Score: 31.986603124542096
```

In [21]:
```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
classify(model, X, y)
```

```
Accuracy: 80.03223856538384
CV Score: 74.50136537757686
```

In [22]:
```
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
classify(model, X, y)
```

```
Accuracy: 88.75680032238566
CV Score: 82.43005492592073
```

In [23]:
```
from sklearn.ensemble import ExtraTreesClassifier
model = ExtraTreesClassifier()
classify(model, X, y)
```

```
Accuracy: 89.09933507958895
CV Score: 83.3266598455934
```

In [24]:
```
import xgboost as xgb
model = xgb.XGBClassifier()
classify(model, X, y)
```

```
Accuracy: 67.62039089260527
CV Score: 64.61326480966859
```

In [25]:
```
import lightgbm
model = lightgbm.LGBMClassifier()
classify(model, X, y)
```

```
Accuracy: 83.80012089462019
CV Score: 78.78828306227706
```

In [ ]: