Red Wine Quality Prediction with Regression Algorithms

Alex Zhukov

10-2022

Table of Contents

Task Formulation

Wine is an alcoholic beverage that has been popular for thousands of years. There are 3 main types of wine - red, white, and rose. These types of wine differ in their base and production processes. Red wine is made by fermenting the juice of dark grapes. During red wine production, the winemaker allows pressed grape juice, called must, to macerate and ferment with the dark grape skins, which adds color, flavor, and tannin to the wine. Alcohol occurs when yeast converts grape sugar into Ethanol and Carbon Dioxide.

Red wine is one of the popular alcoholic drinks nowadays and there is a variety of wine brands out there. So, often people get curious about the quality of red wine. But it becomes a quite complicated task as people don't know how to analyze wine without special knowledge. This special knowledge includes the ability to understand what kind of specific conditions affect the wine quality. Therefore, wine consumers usually rely on ratings and reviews made by a specialist. And sometimes these feedbacks are mutually exclusive and don't help to make the right decision in front of the wine shelves.

To solve this problem, we can use Python to analyze available data. This data can be used to predict the quality of the red wine given certain variables or factors.

Regression Analysis is a set of statistical methods used for the estimation of relationships between a Dependent Variable and one or more Independent Variables. In regression analysis, we used the Independent Variable (x) to estimate the Dependent Variable (y). Both of the variables that will be used must follow these requirements:

- The relationship between the variables is linear
- Both variables must be at least interval scale
- The least-squares criterion is used to determine the equation

There are two types of Regression Analysis: Simple Linear Regression, and Multiple Linear Regression.

Linear Regression Model is a way to model the relationship between two variables

$$\hat{Y} = a + bX$$

where:

- $\hat{Y}$ is the dependent variable (plotted in Y-axis)
- X is the independent variable (plotted in X-axis)
- b is the slope of the line
- a is the y-intercept (constant)

  Multiple Regression a model with multiple (*k*) variables that affect the dependent variable

  $$\hat{Y} = a + b_1X_1 + b_2X_2 + \ldots + b_kX_k$$

  where:

- $X_1 \ldots X_k$ are the independent variables
- a is the y-intercept (constant)
- b_1 - regression coefficient

To define the quality of red wine, there are a lot of factors. It could be the Alcohol (ABV), Sulphates, pH (a measure of how acidic/basic liquid is), etc. That's why we need Multiple Linear Regression here.

So, our Goal:

- **Predict** the perfect ratio of ingredients for red wine. This is a **numerical continuous** outcome.
- **Explore** with various **Regression Models** & see which yields the **greatest accuracy**.
- Examine **trends & correlations** within our data
- Determine which **features** are important in **high quality** Red Wine

Note: *Due to the fact that we are predicting a numerical continuous value, we will be training various Regression Models. Regression analysis is a sub field of Supervised Learning.*

Data Overview

In [1]:
```
import pandas as pd
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

Let's load data. It can be found on Kaggle.

In [2]:
```
df = pd.read_csv('data/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	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5

3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

Our Predictor (Y, Wine Quality) is determined by 11 features (X):

1. ***Fixed acidity (g/L)*** — most acids involved with wine or fixed or nonvolatile (do not evaporate readily)
2. ***Volatile acidity (g/L)*** — amount of acetic acid (which at too high of levels can lead to an unpleasant, vinegar taste)
3. ***Citric acid (mol/L)*** — found in small quantities, citric acid can add ‘freshness’ and flavor to wines
4. ***Residual sugar (g/L)*** — amount of sugar remaining after fermentation stops
5. ***Chlorides (g)*** — amount of salt
6. ***Free sulfur dioxide (mg/L)*** — free form SO^2 exists equilibrium between molecular SO^2 (dissolved gas) and bisulfite ion
7. ***Total sulfur dioxide (mg/L)*** — amount of free and bound forms of SO^2
8. ***Density (g/cm^3)*** — density of water is close to that of water depending on the percent alcohol and sugar content
9. ***pH*** — describes how acidic or basic a wine is on a scale from 0(very acidic) to 14(very basic); most wines are between 3–4
10. ***Sulfates (g)*** — additive which can contribute to sulfur dioxide gas (SO^2) levels, which acts as an antimicrobial
11. ***Alcohol*** — percent alcohol content

We have 1599 observations and 12 variables. One of those is **Quality** which is our target.

In [3]:
```
df.shape
```
Out[3]:
```
(1599, 12)
```

The number of unique values for each variable:

In [4]:
```
df.nunique()
```
Out[4]:
```
fixed acidity            96
volatile acidity        143
citric acid              80
residual sugar           91
chlorides               153
free sulfur dioxide      60
total sulfur dioxide    144
density                 436
pH                       89
sulphates                96
alcohol                  65
quality                   6
dtype: int64
```

Our data has only 1 type of data - **Continuous** - which is **quantitative data** that can be measured

In [5]:
```
df.dtypes
```
Out[5]:
```
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
dtype: object
```

Summarizing the main statistics - the count, mean, standard deviation, min, and max - for our numeric variables.

In [6]:
```
df.describe()
```

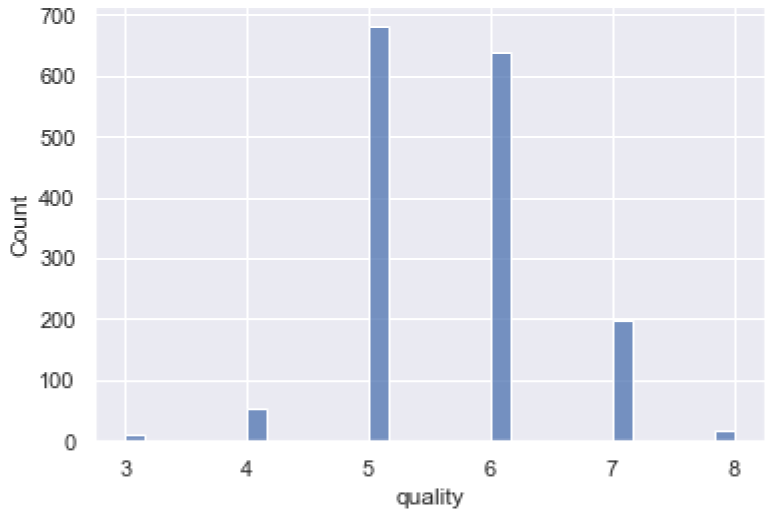
Out[6]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density
count	1599.000000	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	0.996747
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690

The **mean quality** was 5.6, with its **Max** (best quality score) being 8.0 & its **Min** (worst quality score) being 3.0.

Let's take a look on the **quality scores distribution**.

In [7]:
```
sns.histplot(df.quality)
plt.show()
```



Now let's see if we have any missing values we need to take care of.

In [8]:
```
df.isna().sum()
```

Out[8]:
```
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
```

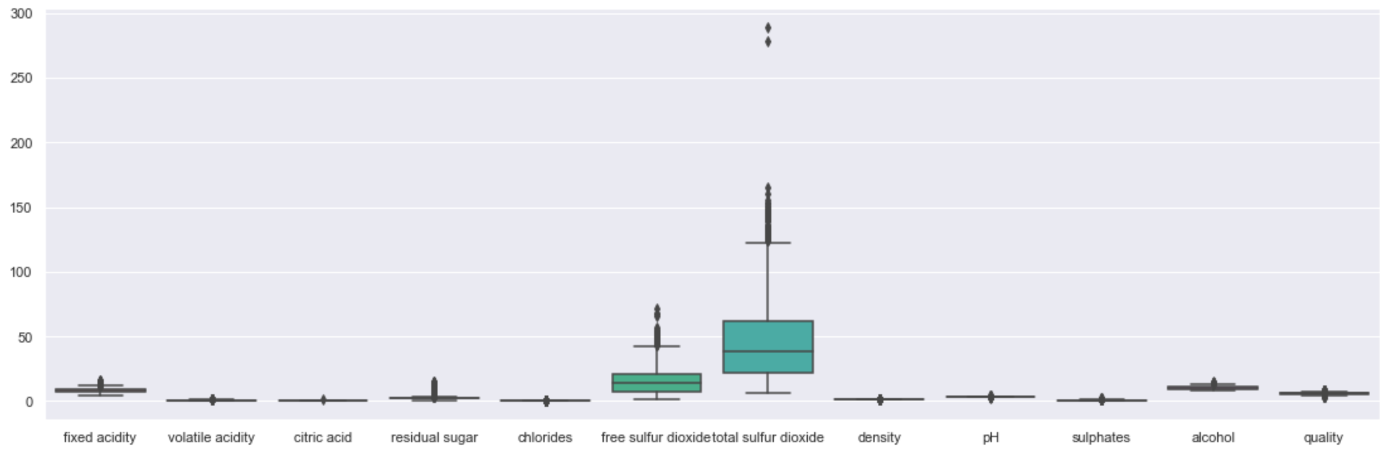
In [9]:
```
df.isna().sum().sum()
```

Out[9]:
```
0
```

No missing values. Great!

But what about **outliers**?

In [10]:
```
plt.figure(figsize=(19,6))
sns.boxplot(data=df)
plt.show()
```



Seems that there are some harmful outliers. Let's statistically detect and remove them in the next step.

Data Preprocessing

I will detect outliers using a **Z-score**.

Z-score *is a measure of* ***position*** *that indicates the number of* ***standard deviations*** *a data value lies from the* ***mean****. Any z-score* ***less*** *thatn -3 or* ***greater*** *thah 3, is an* ***outlier****.*

$$Z = \frac{x - \mu}{\sigma}$$

where:

- x is a score
- μ is the mean
- σ - standard deviation

Actually **99.7%** of our data should be within **3 standard deviations** from the **mean**. All the rest to be removed.

In [11]:
```
z = np.abs(stats.zscore(df)) # obtaining the z-scores of all variables
```

In [12]:
```
z.head()
```

Out[12]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	0.528360	0.961877	1.391472	0.453218	0.243707	0.466193	0.379133	0.558274	1.288643	0.579207	0.960246
1	0.298547	1.967442	1.391472	0.043416	0.223875	0.872638	0.624363	0.028261	0.719933	0.128950	0.584777
2	0.298547	1.297065	1.186070	0.169427	0.096353	0.083669	0.229047	0.134264	0.331177	0.048089	0.584777
3	1.654856	1.384443	1.484154	0.453218	0.264960	0.107592	0.411500	0.664277	0.979104	0.461180	0.584777
4	0.528360	0.961877	1.391472	0.453218	0.243707	0.466193	0.379133	0.558274	1.288643	0.579207	0.960246

Once we calculated the **z-score**, we can remove the outliers to clean our data, by performing the action below.

In [13]:
```
clean_data = df[(z < 3).all(axis=1)]
```

In [14]:
```
df.shape[0] - clean_data.shape[0]
```

Out[14]:
```
148
```

We removed 148 rows which were **outliers**.

So, let's take a look at our new **cleaned up** data.

In [15]:
```
clean_data.describe()
```

Out[15]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density
count	1451.000000	1451.000000	1451.000000	1451.000000	1451.000000	1451.000000	1451.000000	1451.000000
mean	8.310062	0.522950	0.265382	2.387285	0.081425	15.104755	43.735355	0.996710
std	1.646458	0.168531	0.190934	0.862078	0.020966	9.309768	29.441284	0.001716
min	5.000000	0.120000	0.000000	1.200000	0.038000	1.000000	6.000000	0.991500
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	21.000000	0.995600
50%	7.900000	0.520000	0.250000	2.200000	0.079000	13.000000	36.000000	0.996700
75%	9.200000	0.630000	0.420000	2.600000	0.089000	21.000000	58.000000	0.997800
max	13.500000	1.040000	0.790000	6.700000	0.226000	47.000000	145.000000	1.002200

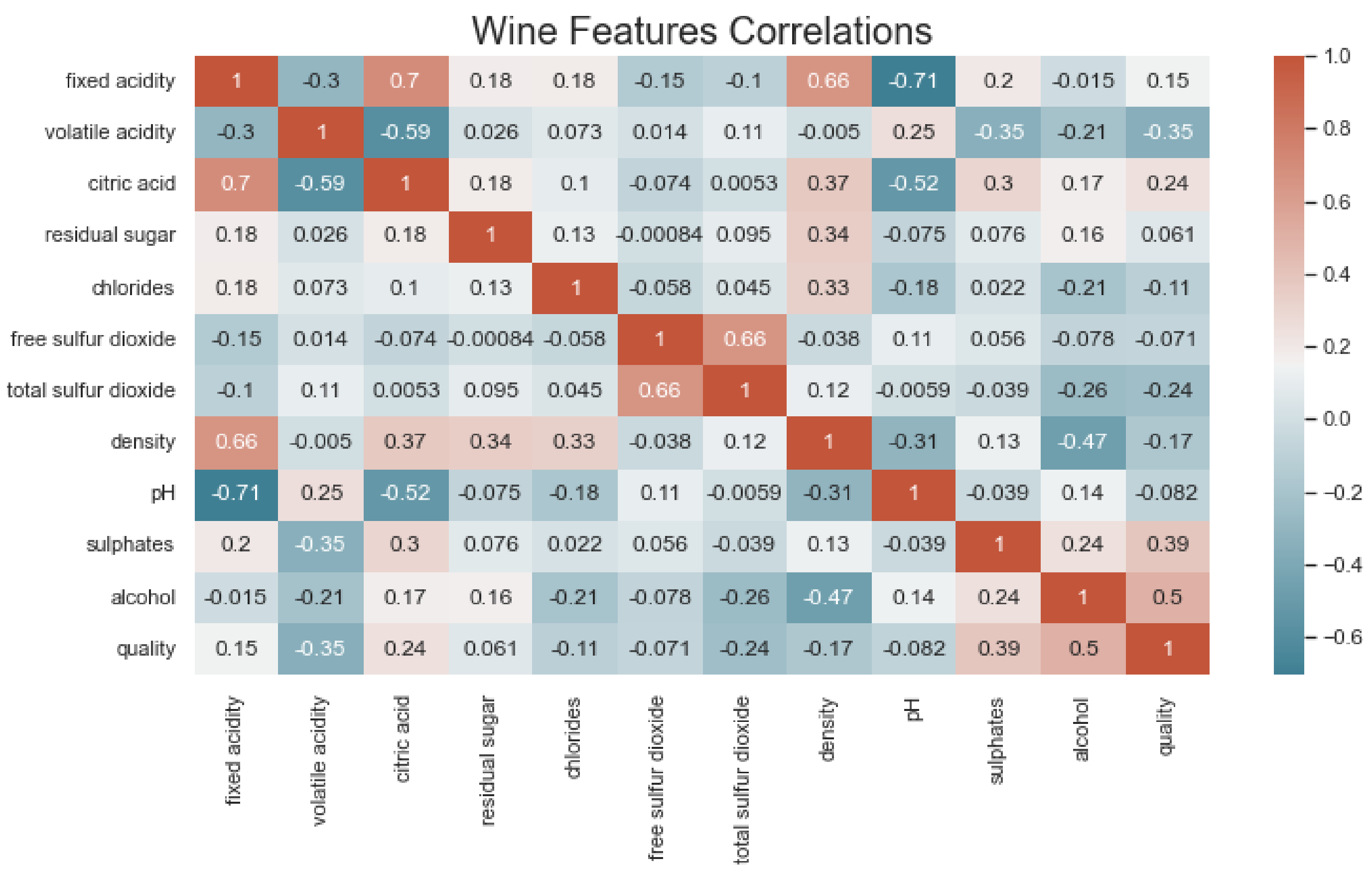
EDA

Correlations

Let's see the **correlations** between all variables with using a **Heat Map**.

In [16]:
```
corr_data = clean_data.corr()
plt.figure(figsize=(12,6))
```

```
fig = sns.heatmap(corr_data, annot=True,
            cmap=sns.diverging_palette(220, 20, as_cmap=True))
plt.title('Wine Features Correlations', fontsize = 20)
#plt.savefig('data/wine_heatmap.png')
plt.show()
```



We can see there is a **strong positive correlation** between **alcohol** and our **predictor**. In fact, this is the most correlated feature in our data set, with a value of 0.5!

Note: *our alcohol feature was the percent alcohol content in a drink. This makes sense that a higher percent of alcohol content would yield a greater satisfaction for a customer purchasing red wine!*

Next, we can see the **second strongest positive correlation**, 0.39, between **Sulphates** and our quality predictor. It seems that people rate the quality higher when an additive (SO^2) is contributed to the drink. Sulphates acts as an antimicrobial.

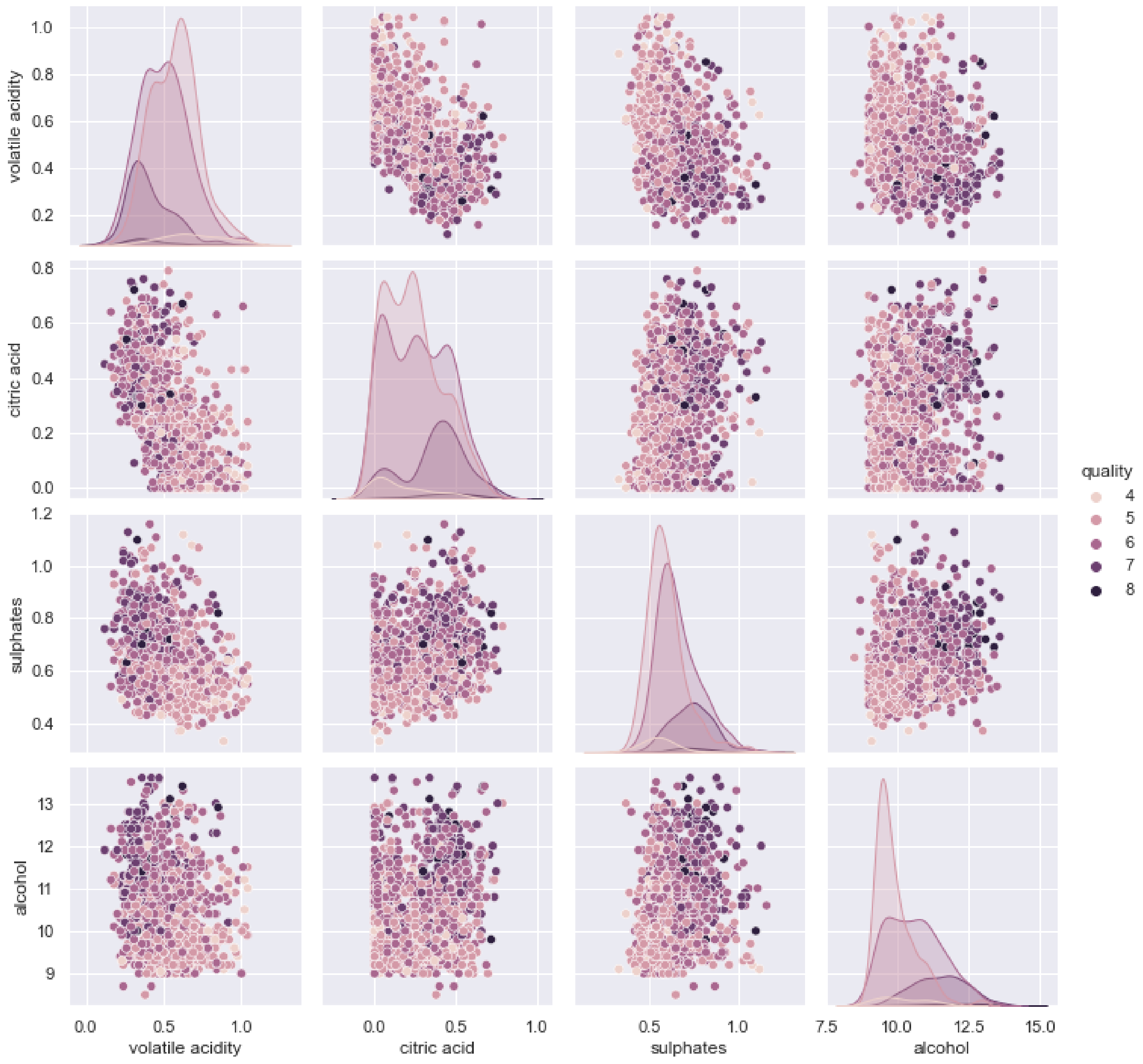
Lastly, the **strongest negative correlation** is the **volatile acidity**, with a correlation of -0.35! This is as expected because too high acetic acid levels can lead to an unpleasant, vinegar taste!

Pair Plots

Pair Plots are also a great way to immediately see the **correlations** between all variables.

Because we have **11 Features**, lets only select **significant features** that **correlate** to our **predictor**, to further examine their **correlation** on a **Pair plot**.

In [17]:
```
significant_data = clean_data[['quality','volatile acidity', 'citric acid', 'sulphates',
sns.pairplot(significant_data, hue='quality')
plt.show()
```



Feature Engineering

We will now do a form of **Feature Engineering**, where we create a new column **classifying** if the wine quality is tasty or not based on its **quality score**!

This new column will be a binary Categorical Data where 0 or 1 indicates if the wine was considered “tasty”.

In [18]:
```
pd.options.mode.chained_assignment = None

tasty = np.where(clean_data['quality'] < 6, 0, 1)
clean_data['tasty'] = tasty
```

In [19]:
```
clean_data.head()
```

Out[19]:

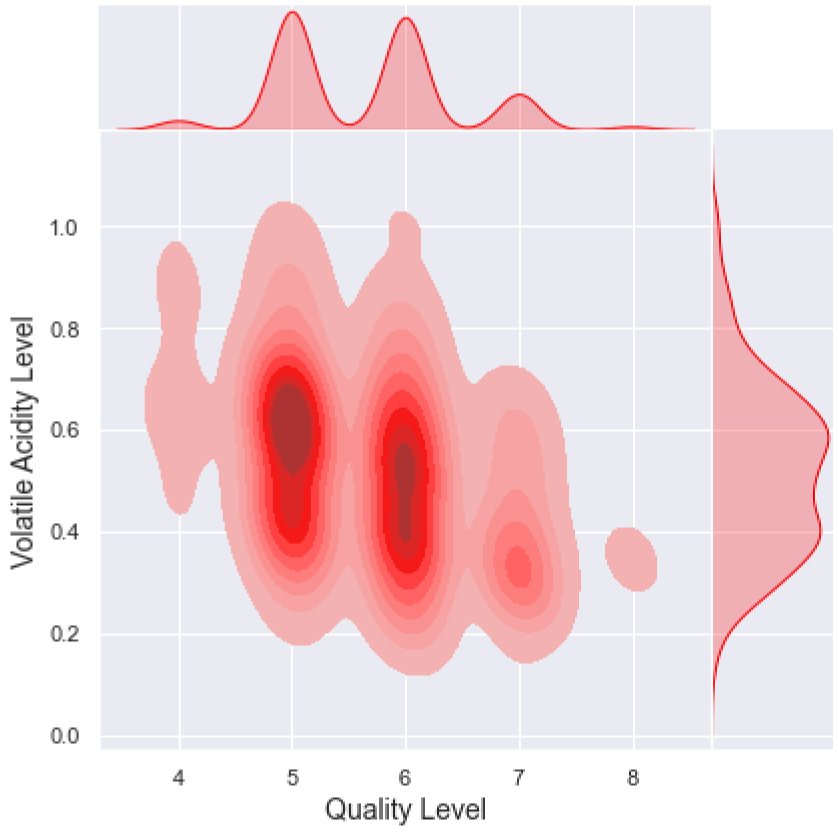
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	tasty
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5	0
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5	0
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5	0

3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6	1
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5	0

Kernel Density Estimation (KDE)

A Kernel Density Estimation (KDE) estimate the probability density function (PDF) of a continuous random variable.

In [20]:
```
g = sns.jointplot(x='quality', y='volatile acidity', data=clean_data, kind='kde', space=
g.set_axis_labels('Quality Level', 'Volatile Acidity Level', fontsize=14)
plt.show()
```



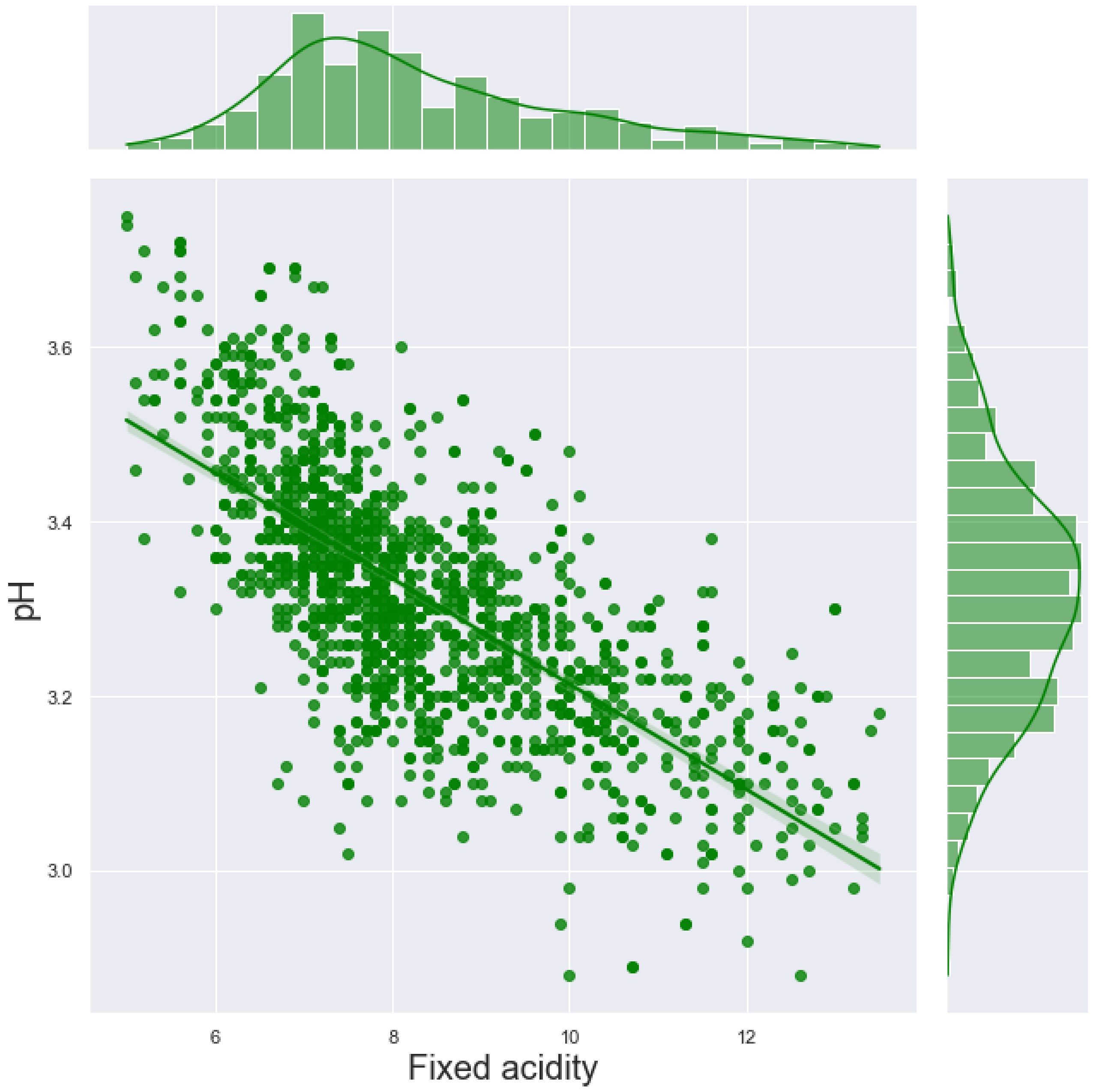
Here we see that **lower quality** red wine is heavily inclined to higher levels of **volatile acidity**. This is because large quantities of acetic acid yield an unpleasant vinegar taste.

Regression Joint Plot

In [21]:
```
g = sns.jointplot(x='fixed acidity',
                  y='pH',
                  data=clean_data,
                  kind='reg',
                  color='green',
                  height=9)

g.set_axis_labels('Fixed acidity', 'pH', fontsize=20)

plt.show()
```

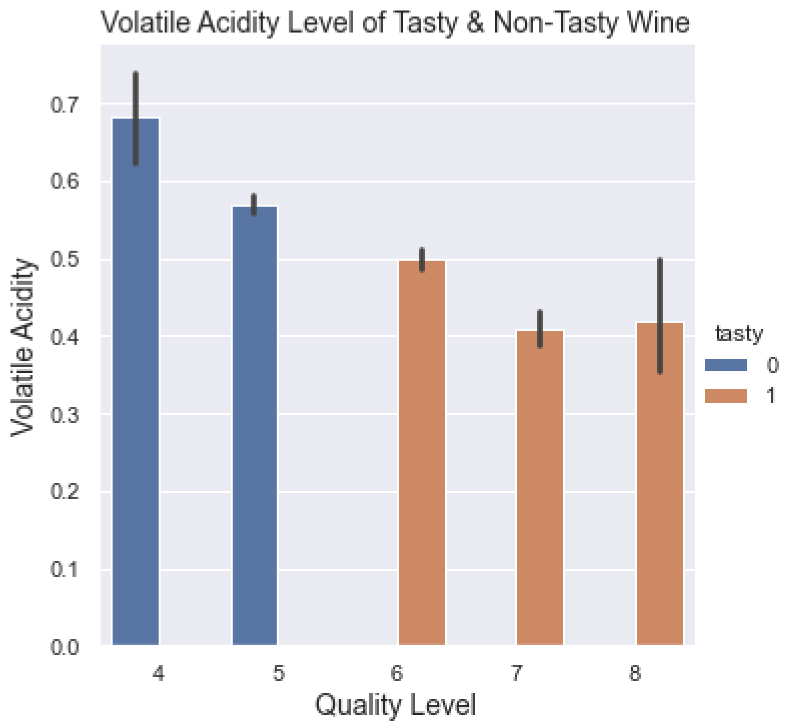


From interpreting the Regression Joint plot above, we can see a **Strong Negative correlation** between **pH** level and **fixed acidity**. In other words, as one of these increases, the other decreases. According to the **HeatMap**, these two features have a **correlation** coefficent of **-0.71**

Bar Plots

In [22]:
```
sns.catplot(x='quality', y='volatile acidity', hue='tasty',
            data=clean_data, kind='bar')
plt.title('Volatile Acidity Level of Tasty & Non-Tasty Wine', fontsize=14)
plt.xlabel('Quality Level', fontsize=14)
plt.ylabel('Volatile Acidity', fontsize=14)

plt.show()
```



This plot again illustrates that high volatile acidity levels yield a terrible tasting wine.

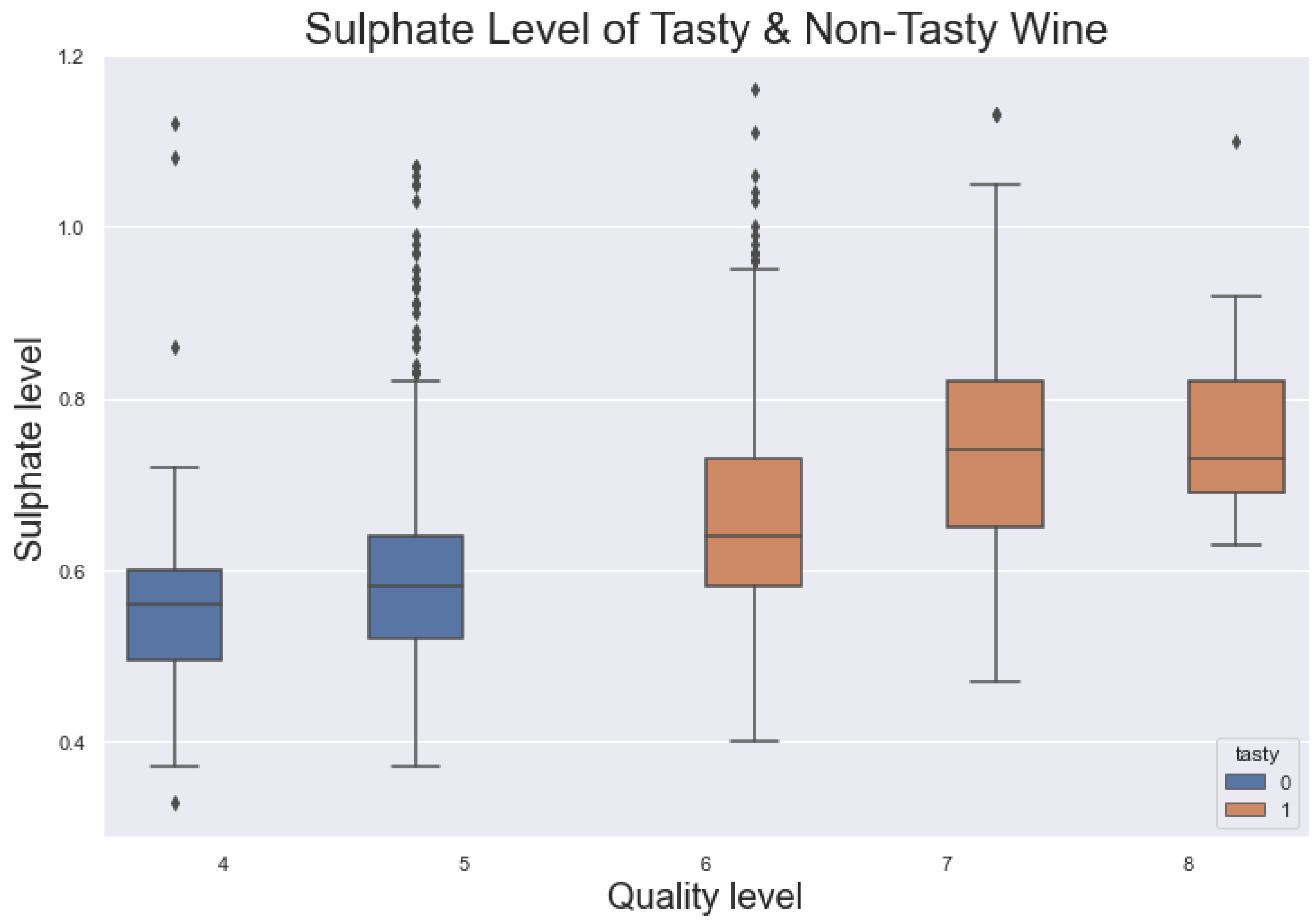
Violin & Box Plots

The advantages of the **Box & Violin** plot is that it shows the **basic statistics** of the data, as well as its **distribution**.

So, let's see the **median**, **IQR**, & **Tukey's fence**. (minimum, first quartile (Q1), median, third quartile (Q3), and maximum).

In [23]:
```
plt.figure(figsize=(12, 8))
sns.boxplot(x='quality', y='sulphates', hue='tasty', data=clean_data)
plt.title('Sulphate Level of Tasty & Non-Tasty Wine', fontsize=24)
plt.xlabel('Quality level', fontsize=20)
plt.ylabel('Sulphate level', fontsize=20)

plt.show()
```

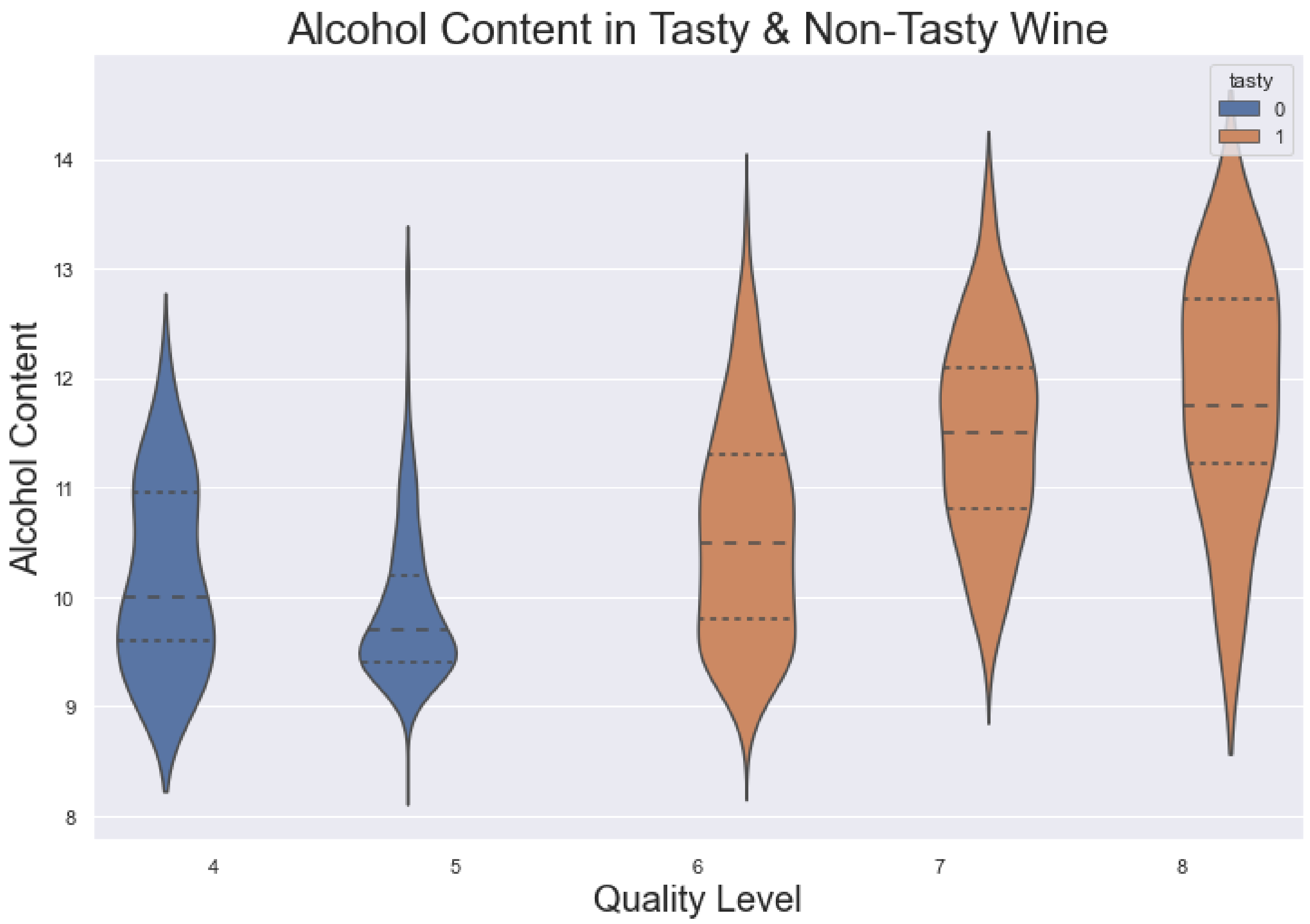


Next, we examine this **box plot** that helps to conclude that a tasty red wine exhibits **heightened median** for **Sulfate** levels.

We can see now that these high sulfate levels would enhance a customers preference.

In [24]:
```
plt.figure(figsize=(12, 8))
sns.violinplot(x='quality', y='alcohol', hue='tasty',
               inner='quartile',
               data=clean_data)
plt.title('Alcohol Content in Tasty & Non-Tasty Wine',fontsize=24)
plt.xlabel('Quality Level', fontsize=20)
plt.ylabel('Alcohol Content', fontsize=20)

plt.show()
```



After analyzing this **violin plot** we can conclude that the overall shape & distribution for tasty & non-tasty wine **differ vastly**. **Tasty** red wine exhibits a **higher median** for percent **alcohol** content & thus a great distribution of it is between 10 and 13, while **non-tasty** red wine consists of a **lower median alcohol** level content between 9.5 and 11.

Let's now compare the **averages** between **tasty** and **non-tasty** red wine.

```
In [25]: tasty = clean_data[clean_data['tasty'] == 1]
         tasty.describe()
```

Out[25]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
count	787.000000	787.000000	787.000000	787.000000	787.000000	787.000000	787.000000	787.000000	787.00000
mean	8.485642	0.477541	0.295286	2.400762	0.079785	14.560356	36.626429	0.996481	3.31186
std	1.748845	0.162388	0.197117	0.871710	0.022654	9.061416	21.715608	0.001856	0.13703
min	5.100000	0.120000	0.000000	1.200000	0.038000	1.000000	6.000000	0.991500	2.88000
25%	7.200000	0.360000	0.105000	1.900000	0.067000	7.000000	20.000000	0.995220	3.22000
50%	8.100000	0.460000	0.310000	2.200000	0.077000	13.000000	31.000000	0.996400	3.31000
75%	9.650000	0.582500	0.450000	2.550000	0.087000	19.000000	48.000000	0.997600	3.40000
max	13.400000	1.040000	0.760000	6.700000	0.226000	45.000000	114.000000	1.002200	3.71000

```
In [26]: nontasty = clean_data[clean_data['tasty'] == 0]
         nontasty.describe()
```

Out[26]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
count	664.000000	664.000000	664.000000	664.000000	664.000000	664.000000	664.000000	664.000000	664.00000
mean	8.101958	0.576770	0.229940	2.371310	0.083369	15.750000	52.161145	0.996981	3.32075
std	1.490911	0.159709	0.177049	0.850899	0.018601	9.562567	34.721033	0.001489	0.14572
min	5.000000	0.180000	0.000000	1.200000	0.039000	3.000000	6.000000	0.992560	2.88000
25%	7.100000	0.460000	0.080000	1.900000	0.074000	8.000000	23.000000	0.996100	3.22000
50%	7.800000	0.580000	0.220000	2.200000	0.080000	14.000000	43.000000	0.996900	3.32000
75%	8.825000	0.670000	0.350000	2.600000	0.090000	21.250000	72.000000	0.997823	3.41000
max	13.500000	1.040000	0.790000	6.600000	0.186000	47.000000	145.000000	1.001800	3.75000

In [27]:
```
print(f'Tasty Wine Sulphates Level: {tasty["sulphates"].mean()}')
print(f'Non-Tasty Wine Sulphates Level: {nontasty["sulphates"].mean()}')
print()
print(f'Tasty Wine Alcohol Content Level: {tasty["alcohol"].mean()}')
print(f'Non-Tasty Wine Alcohol Content Level: {nontasty["alcohol"].mean()}')
print()
print(f'Tasty Wine Total Sulfur Dioxide level: {tasty["total sulfur dioxide"].mean()}')
print(f'Non-Tasty Wine Total Sulfur Dioxide level: {nontasty["total sulfur dioxide"].mea
```

```
Tasty Wine Sulphates Level: 0.6825794155019054
Non-Tasty Wine Sulphates Level: 0.5951807228915662

Tasty Wine Alcohol Content Level: 10.833989834815748
Non-Tasty Wine Alcohol Content Level: 9.931701807228912

Tasty Wine Total Sulfur Dioxide level: 36.62642947903431
Non-Tasty Wine Total Sulfur Dioxide level: 52.161144578313255
```

Tastier Wine exhibits **higher** levels of **Sulphates** and **Alcohol** and **lower** level of Total Sulfur Dioxid.

Therefore, to maximize the sales of red wine, wine producer need to include low levels of **sulfur dioxide** and **high** levels of **sulphates** and **alcohol**.

Predictive Analysis

Prepare Data for Modeling

Assign the 11 features to X, and the `quality` column to our predictor - y

In [28]:
```
X = clean_data.drop(['quality', 'tasty'], axis=1)
y = clean_data.quality
```

Split the dataset into the Training set and Test set

In [29]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=.2, random_state=1)
```

Normalizing the data - transforming it so that its distribution will have a mean of 0 and a standard deviation of 1.

In [30]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

Modeling/Training

Now I'm going to train various **Regression Models** on the **Training set** and see which yields the **highest accuracy**. I will compare the accuracy of **Multiple Linear Regression, Ridge Regression, SVR (Support Vector Regression), Decision Tree Regression, Random Forest Regression, and XGBoost.** These are **supervised learning** models, for predicting continuous values.

Note: *There are a few metrics for measuring accuracy for a regression model like* ***Root Mean Squared Error (RMSE), Residual Standard Error (RSE),*** *R^2***, and*** *Mean Squared Error (MSE)*___.*** But I will be measuring our models with*** *Mean Absolute Error (MAE)*__

In [31]:
```
from sklearn.metrics import mean_absolute_error, r2_score, accuracy_score
```

Model 1: Multiple Linear Regression

In [32]:
```
from sklearn.linear_model import LinearRegression

# Train model on the train dataset
reg1 = LinearRegression()
reg1.fit(x_train, y_train)

# predicting the test dataset
y_pred1 = reg1.predict(x_test)

# Evaluating model performance
print(f'MAE: {mean_absolute_error(y_test, y_pred1)}\n \
      Train data r2 score: {reg1.score(x_train, y_train)}\n \
      Test data r2 score: {r2_score(y_test, y_pred1)}')
```

```
MAE: 0.4939537644033682
       Train data r2 score: 0.40505473059191754
       Test data r2 score: 0.25451438574191876
```

Model 2: Ridge Regression

In [33]:
```
from sklearn.linear_model import Ridge
reg2 = Ridge()
reg2.fit(x_train, y_train)
y_pred2 = reg2.predict(x_test)

print(f'MAE: {mean_absolute_error(y_test, y_pred2)}\n \
      Train data r2 score: {reg2.score(x_train, y_train)}\n \
      Test data r2 score: {r2_score(y_test, y_pred2)}')
```

```
MAE: 0.4939240146691417
       Train data r2 score: 0.40505434091133596
       Test data r2 score: 0.2546478172026633
```

Model 3: Support Vector Regression

In [34]:
```
from sklearn.svm import SVR
reg3 = SVR()
reg3.fit(x_train, y_train)
y_pred3 = reg3.predict(x_test)
```

```
print(f'MAE: {mean_absolute_error(y_test, y_pred3)}\n \
      Train data r2 score: {reg3.score(x_train, y_train)}\n \
      Test data r2 score: {r2_score(y_test, y_pred3)}')
```

```
MAE: 0.45410837591861164
       Train data r2 score: 0.6070471406274619
       Test data r2 score: 0.2904305638776502
```

Model 4: Decision Tree Regression

In [35]:
```
from sklearn.tree import DecisionTreeRegressor
reg4 = DecisionTreeRegressor(criterion='poisson', max_depth=3, max_leaf_nodes=4)
reg4.fit(x_train, y_train)
y_pred4 = reg4.predict(x_test)

print(f'MAE: {mean_absolute_error(y_test, y_pred4)}\n \
      Train data r2 score: {reg4.score(x_train, y_train)}\n \
      Test data r2 score: {r2_score(y_test, y_pred4)}')
```

```
MAE: 0.5433287304626896
       Train data r2 score: 0.2969643822023026
       Test data r2 score: 0.18116070615480218
```

Model 5: Random Forest Regression

In [36]:
```
from sklearn.ensemble import RandomForestRegressor
reg5 = RandomForestRegressor(n_estimators = 200, max_depth=13 , random_state=0)
reg5.fit(x_train, y_train)
y_pred5= reg5.predict(x_test)

print(f'MAE: {mean_absolute_error(y_test, y_pred5)}\n \
      Train data r2 score: {reg5.score(x_train, y_train)}\n \
      Test data r2 score: {r2_score(y_test, y_pred5)}')
```

```
MAE: 0.4226459192249129
       Train data r2 score: 0.916430096255789
       Test data r2 score: 0.3687445367908615
```

Model 6: Gradient Boost Regression

In [37]:
```
from xgboost import XGBRegressor
reg6 = XGBRegressor()
reg6.fit(x_train, y_train)
y_pred6 = reg6.predict(x_test)

print(f'MAE: {mean_absolute_error(y_test, y_pred6)}\n \
      Train data r2 score: {reg6.score(x_train, y_train)}\n \
      Test data r2 score: {r2_score(y_test, y_pred6)}')
```

```
MAE: 0.4055099143195398
       Train data r2 score: 0.9970469089757216
       Test data r2 score: 0.3386095956511733
```

Model 7: K-Neighbors Regression

In [38]:
```
from sklearn.neighbors import KNeighborsRegressor
reg7 = KNeighborsRegressor()
reg7.fit(x_train, y_train)
y_pred7 = reg7.predict(x_test)

print(f'MAE: {mean_absolute_error(y_test, y_pred7)}\n \
      Train data r2 score: {reg7.score(x_train, y_train)}\n \
      Test data r2 score: {r2_score(y_test, y_pred7)}')
```

```
MAE: 0.5429553264604811
       Train data r2 score: 0.563044328015119
       Test data r2 score: 0.08433500021011031
```

Among all these models, XGboost shows the most accurate result.

Therefore I will try to tune this model to improve its accuracy. Best parameters for the model can be made found using **GridSearchCV**.

In [39]:
```
from sklearn.model_selection import GridSearchCV

xgb_reg = XGBRegressor()

params = {     'learning_rate': [.03, 0.05, .07],
               'max_depth': [12, 14, 16],
               'min_child_weight': [4],
               'subsample': [0.7],
               'colsample_bytree': [0.7],
               'n_estimators': [500]
         }
grid_xgb_reg=GridSearchCV(xgb_reg,
                          param_grid=params,
                          cv=3,
                          n_jobs=-1)

grid_xgb_reg.fit(x_train, y_train)
best_model = grid_xgb_reg.best_estimator_
y_pred11 = best_model.predict(x_test)
```

In [40]:
```
print(f'MAE: {mean_absolute_error(y_test, y_pred11)}\n \
      Train data r2 score: {grid_xgb_reg.score(x_train, y_train)}\n \
      Test data r2 score: {r2_score(y_test, y_pred11)}')
```

```
MAE: 0.38015904869000094
       Train data r2 score: 0.999922648571115
       Test data r2 score: 0.3803333214790495
```

Conclusion

We found that XGBoost Regression is the best model for our task that provides 0.38 MAE which is relatively good result. Having our target variable range from 4 to 8 we have an average error of 0.38.

However, there's one **important point** that we have to keep in mind.

In this project we considered our predictor as a **continuous numerical target**. And that is not 100% correct. The score of the red wine can be either 4, 5, 6, 7 or 8. And this is more likely a **nominal** variable with **5 categories** (or even **ordinal**) but not **continuous** as we assumed earlier in this project. Therefore, when we apply **linear regression** models we got continuous results in the range between 4 and 8. Of course we still can evaluate the red wine quality based on the floating number score we got, but the correct way would be to perform **multinominal logistic regression** or **classification** techniques in this task. And these models will work with even better result.

Let's see just one example below.

In [41]:
```
# convert our target into categories from 0 to 4
y_train_log = y_train-4
y_test_log = y_test-4
```

In [42]:
```
# apply gradient boost classifier
from xgboost import XGBClassifier

clf = XGBClassifier()
clf.fit(x_train, y_train_log)

y_pred_log = clf.predict(x_test)
```

In [43]:
```
print(f'MAE: {mean_absolute_error(y_test_log, y_pred_log)}\nAccuracy score: {accuracy_sc
```

```
MAE: 0.28865979381443296
Accuracy score: 0.7491408934707904
```

And finally let's look at confusion matrix

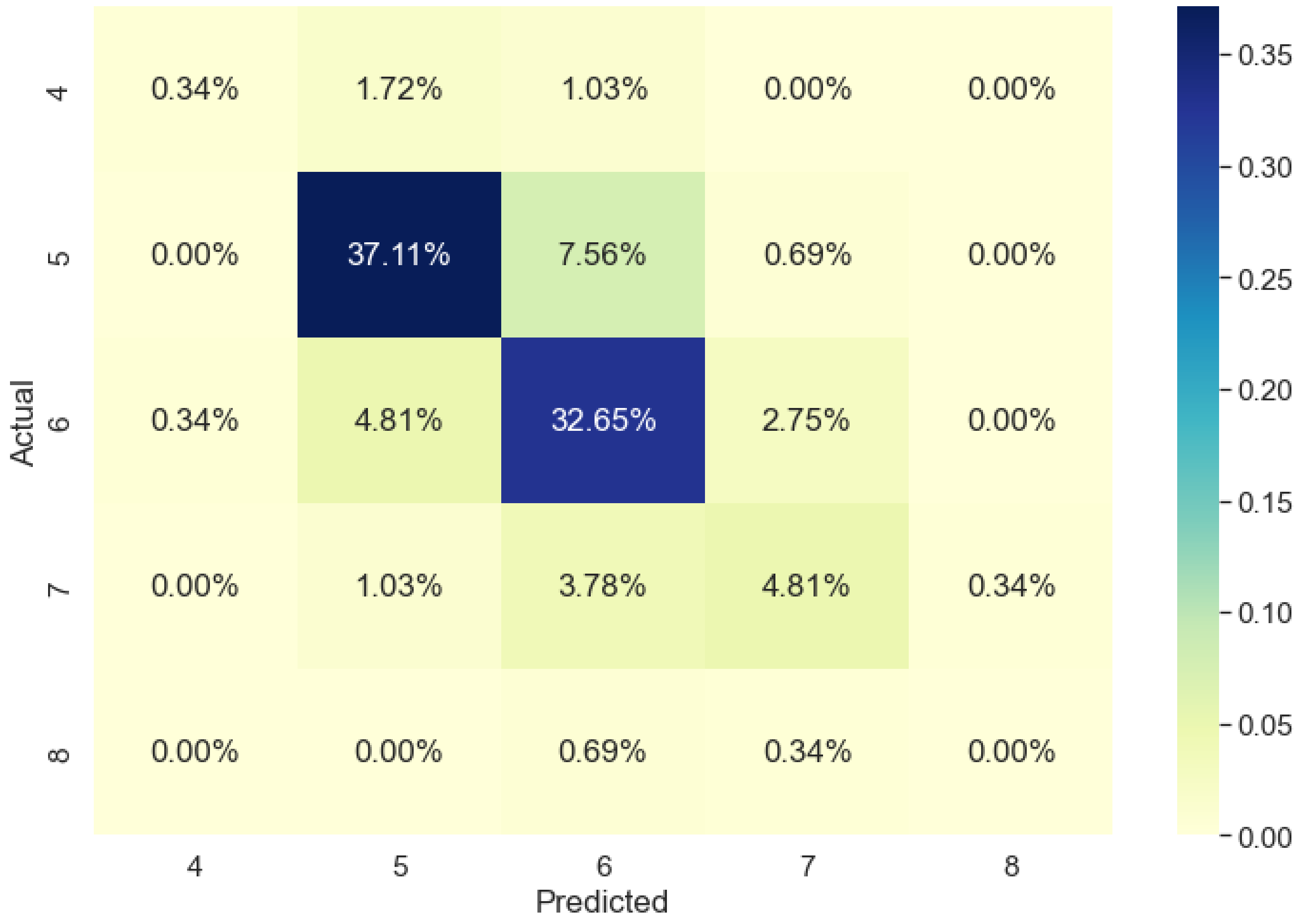
In [44]:
```
from sklearn.metrics import confusion_matrix, plot_confusion_matrix
cf = confusion_matrix(y_test_log, y_pred_log)

cf
```

Out[44]:
```
array([[  1,   5,   3,   0,   0],
       [  0, 108,  22,   2,   0],
       [  1,  14,  95,   8,   0],
       [  0,   3,  11,  14,   1],
       [  0,   0,   2,   1,   0]], dtype=int64)
```

In [45]:
```
df_cm = pd.DataFrame(cf, columns=np.unique(y_test), index = np.unique(y_test))
df_cm.index.name = 'Actual'
df_cm.columns.name = 'Predicted'

plt.figure(figsize = (12,8))
sns.set(font_scale=1.4)
sns.heatmap(df_cm/np.sum(cf), cmap="YlGnBu",  fmt='.2%', annot=True)
plt.show()
```


Actual
Predicted
4
5
6
7
8
0.34% 1.72% 1.03% 0.00% 0.00%
0.00% 37.11% 7.56% 0.69% 0.00%
0.34% 4.81% 32.65% 2.75% 0.00%
0.00% 1.03% 3.78% 4.81% 0.34%
0.00% 0.00% 0.69% 0.34% 0.00%
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00