Prediction Wine Quality

The quality of a wine is determined by 11 input variables:

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)Sulfates

11)Alcohol

Setup

First, I imported all of the relevant libraries that I'll be using as well as the data itself.

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
import seaborn as sns
import plotly.express as px
```

Reading Data

In [2]:

```
data = pd.read_csv("Desktop/winequality_red.csv")
data.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

Understanding Data

This is a very beginner-friendly dataset. I did not have to deal with any missing values, and there isn't much flexibility to conduct some feature engineering given these variables. Next, I wanted to explore my data a little bit more.

In [3]:

```
data.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
```

This gives a concise summary of the dataframe. We can see there are no null values in the dataframe, including 12 columns and 1599 entries.The dataset is already clean and tidy. In total there are 12 column where last column represents quality and rest are properties related to wines.

In [4]:

```
data.describe()
```

Out[4]:

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

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
count	1599.000000	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	3.311113
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000

This is used to view some basic statistical details like percentile, mean, std etc. of a data frame or a series of numeric values.

In [13]:

```
fig = px.histogram(data,x='quality')
print(fig.show())
```

None

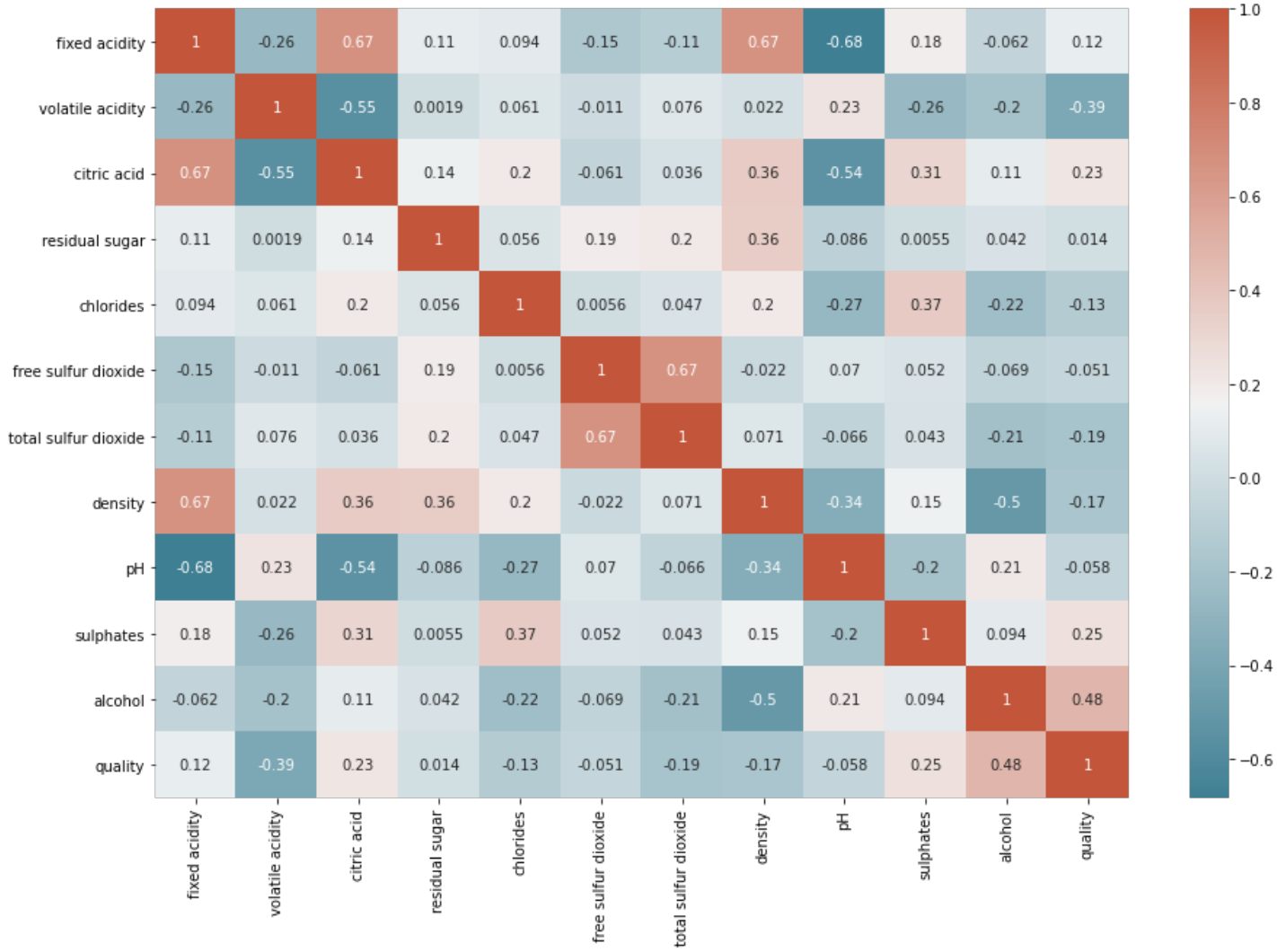
Correlation Matrix

In [6]:

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

Out[6]:

<AxesSubplot:>



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'.

I separated my feature variables (X) and the target variable (y) into separate dataframes.

In [7]:

```
# Create Classification version of target variable
data['goodquality'] = [1 if x >= 7 else 0 for x in data['quality']]
# Separate feature variables and target variable
X = data.drop(['quality','goodquality'], axis = 1)
y = data['goodquality']
```

Proportion of Good vs Bad Wines

In [8]:

```
# See proportion of good vs bad wines
data['goodquality'].value_counts()
```

Out[8]:

```
0    1382
1     217
Name: goodquality, dtype: int64
```

Preparing Data for Modelling

In [9]:

```
# Normalize feature variables
from sklearn.preprocessing import StandardScaler
```

```
X_features = X
X = StandardScaler().fit_transform(X)
```

Split data

In [10]:

```
# Splitting the data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.25, random_state=0)
```

Modelling DecisionTree

In [11]:

```
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier
model1 = DecisionTreeClassifier(random_state=1)
model1.fit(X_train, y_train)
y_pred1 = model1.predict(X_test)
print(classification_report(y_test, y_pred1))
```

```
              precision    recall  f1-score   support

           0       0.96      0.92      0.94       355
           1       0.53      0.73      0.62        45

    accuracy                           0.90       400
   macro avg       0.75      0.83      0.78       400
weighted avg       0.92      0.90      0.90       400
```

Random Forest

In [12]:

```
from sklearn.ensemble import RandomForestClassifier
model2 = RandomForestClassifier(random_state=1)
model2.fit(X_train, y_train)
y_pred2 = model2.predict(X_test)
print(classification_report(y_test, y_pred2))
```

```
              precision    recall  f1-score   support

           0       0.95      0.97      0.96       355
           1       0.68      0.58      0.63        45

    accuracy                           0.92       400
   macro avg       0.82      0.77      0.79       400
weighted avg       0.92      0.92      0.92       400
```

In [ ]:

```

```