Wine Quality Prediction Using ML algorithms

Import Libraries

****Import the usual libraries for pandas and plotting.**

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib as plt
import seaborn as sns
import plotly.express as px
%matplotlib inline
```

Reading Data

In [2]:

```
wine=pd.read_csv('winequality-red.csv', delimiter=';', skiprows=0, low_memory=False)
```

Descriptive statistics

For the next step, we have to check what technical information contained in the data,

In [3]:

```
wine.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
fixed acidity           1599 non-null float64
volatile acidity        1599 non-null float64
citric acid             1599 non-null float64
residual sugar          1599 non-null float64
chlorides               1599 non-null float64
free sulfur dioxide     1599 non-null float64
total sulfur dioxide    1599 non-null float64
density                 1599 non-null float64
pH                      1599 non-null float64
sulphates               1599 non-null float64
alcohol                 1599 non-null float64
quality                 1599 non-null int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
```

In [4]:

```
wine.head()
```

Out[4]:

	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

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

In [5]:

```
wine.describe()
```

Out[5]:

	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

In [6]:

```
wine.isna().any()
```

Out[6]:

```
fixed acidity           False
volatile acidity        False
citric acid             False
residual sugar          False
chlorides               False
free sulfur dioxide     False
total sulfur dioxide    False
density                 False
pH                      False
sulphates               False
alcohol                 False
quality                 False
dtype: bool
```

In [7]:

```
wine.isnull().sum()
```

Out[7]:

```
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
```

Comment: There are no null and NA values, so we dont need to deal with any dataset changes

Distribution of the dependent variable

In this case, depedent variable is "quality"

In [8]:

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

Correalation

**** We want to see the correlations between the variables, so we could get a much better understanding of the relationships between them**

In [9]:

```
corr = wine.corr()
corr
```

Out[9]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	q
fixed acidity	1.000000	-0.256131	0.671703	0.114777	0.093705	-0.153794	-0.113181	0.668047	-0.682978	0.183006	-0.061668	0.1
volatile acidity	-0.256131	1.000000	-0.552496	0.001918	0.061298	-0.010504	0.076470	0.022026	0.234937	-0.260987	-0.202288	0.3
citric acid	0.671703	-0.552496	1.000000	0.143577	0.203823	-0.060978	0.035533	0.364947	-0.541904	0.312770	0.109903	0.2
residual sugar	0.114777	0.001918	0.143577	1.000000	0.055610	0.187049	0.203028	0.355283	-0.085652	0.005527	0.042075	0.0
chlorides	0.093705	0.061298	0.203823	0.055610	1.000000	0.005562	0.047400	0.200632	-0.265026	0.371260	-0.221141	0.1

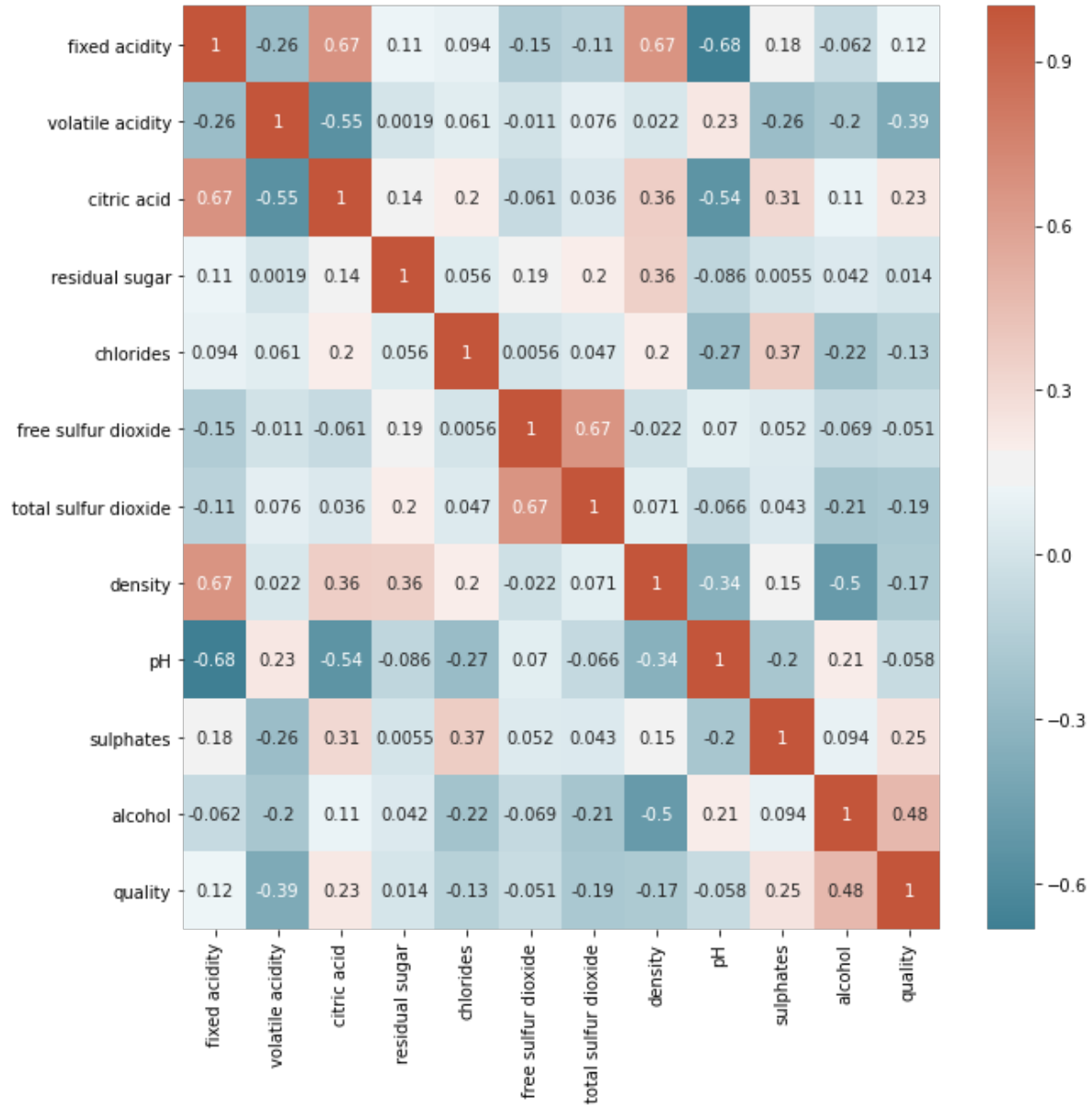
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	
free sulfur dioxide	-0.153794	-0.010504	-0.060978	0.187049	0.005562	1.000000	0.667666	-0.021946	0.070377	0.051658	-0.069408	0.0
total sulfur dioxide	-0.113181	0.076470	0.035533	0.203028	0.047400	0.667666	1.000000	0.071269	-0.066495	0.042947	-0.205654	0.1
density	0.668047	0.022026	0.364947	0.355283	0.200632	-0.021946	0.071269	1.000000	-0.341699	0.148506	-0.496180	0.1
pH	-0.682978	0.234937	-0.541904	-0.085652	-0.265026	0.070377	-0.066495	-0.341699	1.000000	-0.196648	0.205633	0.0
sulphates	0.183006	-0.260987	0.312770	0.005527	0.371260	0.051658	0.042947	0.148506	-0.196648	1.000000	0.093595	0.2
alcohol	-0.061668	-0.202288	0.109903	0.042075	-0.221141	-0.069408	-0.205654	-0.496180	0.205633	0.093595	1.000000	0.4
quality	0.124052	-0.390558	0.226373	0.013732	-0.128907	-0.050656	-0.185100	-0.174919	-0.057731	0.251397	0.476166	1.0

In [10]:

```
import matplotlib.pyplot as plts
fig, ax = plts.subplots(figsize=(10,10))
sns.heatmap(corr, xticklabels=corr.columns, yticklabels=corr.columns, annot=True, cmap=s
ns.diverging_palette(220, 20, as_cmap=True))
```

Out[10]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1fba7155548>
```



Now, we have to find those features that are fully correlated to each other, so we can reduce the number of features from the data.

If the correlation number (feature) has value above 0.7 it was considered as a fully correlated feature, so we drop that feature.

In [11]:

```
for a in range(len(wine.corr().columns)):
    for b in range(a):
        if abs(wine.corr().iloc[a,b]) >0.7:
            name = wine.corr().columns[a]
            print(name)
```

Comment: In this dataset, we dont need to drop variables

Convert to a Classification Problem

We want to convert dependent variable "quality" into binary output "goodquality", where we define a bottle of wine as 'good quality' if it has a quality score of 7 or higher, which is equal to ONE, or else it is a 'bad quality', which is equal to ZERO.

In [12]:

```
# Create Classification version of target variable
wine['goodquality'] = [1 if x >= 7 else 0 for x in wine['quality']]
```

In [13]:

```
wine.head()
```

Out[13]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	goodquality
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	0
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

In [14]:

```
wine['goodquality'].value_counts()
```

Out[14]:

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

Preparing Data for Modelling

We separate feature variables (X) and the target variable (y) into separate dataframes

In [15]:

```
X = wine.drop(['quality','goodquality'], axis = 1)
y = wine['goodquality']
```

In [16]:

```
X.head()
```

Out[16]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
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 [17]:

```
y.value_counts()
```

Out[17]:

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

Normalize feature variables

Now, we standardize the data, which means that it will transform the data so that its distribution will have a mean of 0 and a standard deviation of 1. It is important to standardize data in order to equalize the range of the data.

In [18]:

```
from sklearn.preprocessing import StandardScaler
X_features = X
X = StandardScaler().fit_transform(X)
```

Splitting the data, on train set and test set

In [19]:

```
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)
```

Finding the best ML model

Model 1: Decision Tree

In [20]:

```
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier
model1 = DecisionTreeClassifier(random_state=1)
model1.fit(X_train, y_train)
# predicting score
model1_score = model1.score(X_test,y_test)
print('score of model is : ',model1_score)
y_pred1 = model1.predict(X_test)
print(classification_report(y_test, y_pred1))
```

```
score of model is :  0.8975
              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
```

Model 2: Random Forest

In [21]:

```
from sklearn.ensemble import RandomForestClassifier
model2 = RandomForestClassifier(random_state=1)
model2.fit(X_train, y_train)
# predicting score
model2_score = model2.score(X_test,y_test)
print('score of model is : ',model2_score)
y_pred2 = model2.predict(X_test)
print(classification_report(y_test, y_pred2))
```

```
score of model is :  0.9225
              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
```

Model 3: AdaBoost

In [22]:

```
from sklearn.ensemble import AdaBoostClassifier
model3 = AdaBoostClassifier(random_state=1)
model3.fit(X_train, y_train)
# predicting score
model3_score = model3.score(X_test,y_test)
print('score of model is : ',model3_score)
y_pred3 = model3.predict(X_test)
print(classification_report(y_test, y_pred3))
```

```
score of model is :  0.89
              precision    recall  f1-score   support

           0       0.94      0.94      0.94       355
           1       0.51      0.49      0.50        45

    accuracy                           0.89       400
   macro avg       0.72      0.71      0.72       400
weighted avg       0.89      0.89      0.89       400
```

Model 4: Gradient Boosting

In [23]:

```
from sklearn.ensemble import GradientBoostingClassifier
model4 = GradientBoostingClassifier(random_state=1)
model4.fit(X_train, y_train)
# predicting score
model4_score = model4.score(X_test,y_test)
print('score of model is : ',model4_score)
y_pred4 = model4.predict(X_test)
```

```
print(classification_report(y_test, y_pred4))
```

```
score of model is :  0.8925
              precision    recall  f1-score   support

           0       0.94      0.94      0.94       355
           1       0.52      0.51      0.52        45

    accuracy                           0.89       400
   macro avg       0.73      0.73      0.73       400
weighted avg       0.89      0.89      0.89       400
```

CONCLUSION:By comparing the four models, the random forest has the highest level of accuracy(score of model). Also, that model has the best f1-score for predicting good quality wines (1), which value is 0.63. So, we can conclude, the RANDOM FOREST is the best model.

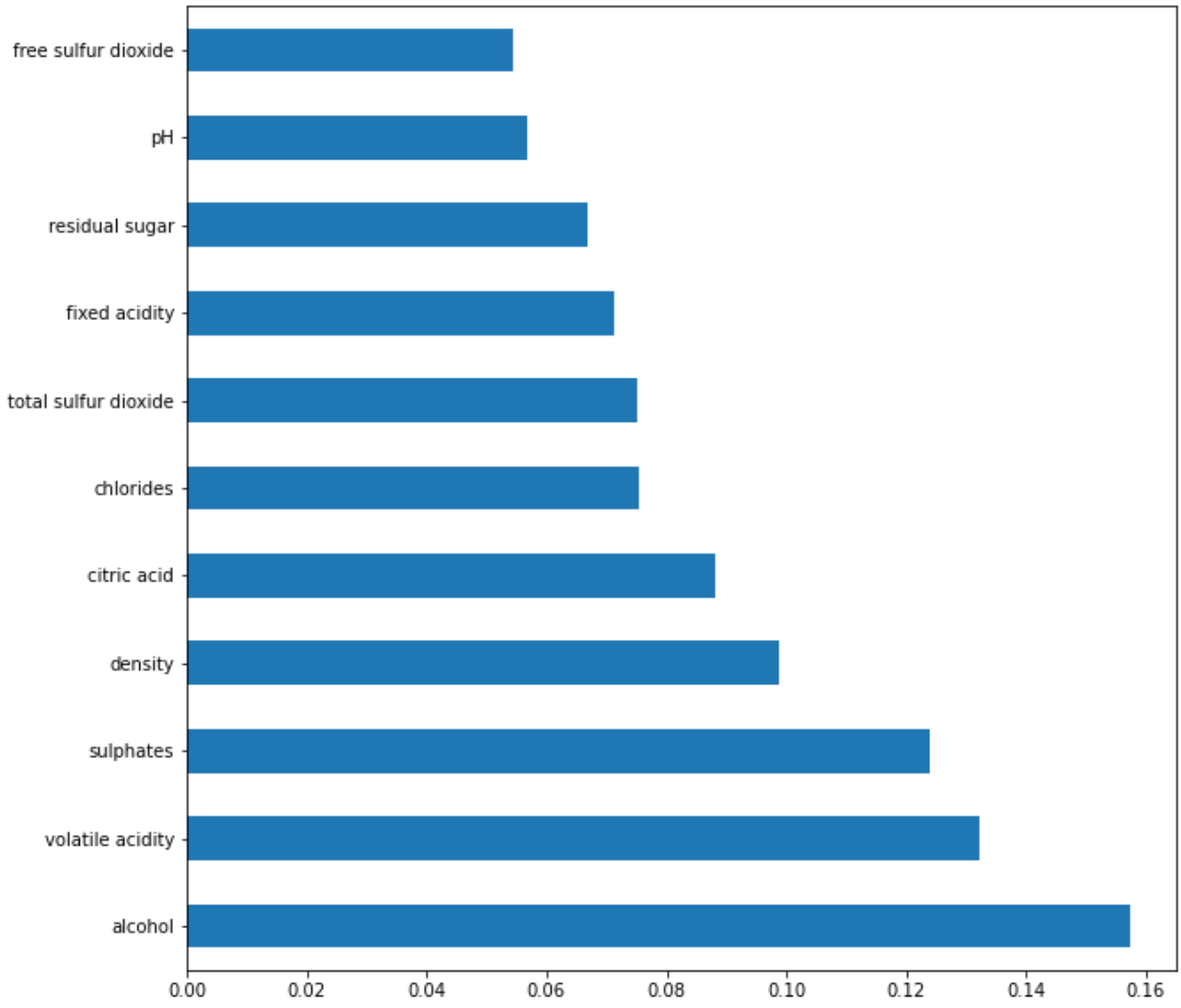
Feature Importance

In [24]:

```
feat_importances = pd.Series(model2.feature_importances_, index=X_features.columns)
feat_importances.nlargest(25).plot(kind='barh',figsize=(10,10))
```

Out[24]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1fba30315c8>
```



As we can see from a graphic above, the top 3 importance features based on the Random Forest are: alcohol, volatile acidity, and sulphates.

Comparing the Top 4 Features

We will split the dataset into good quality and bad quality to compare averages of these variables for more details.

In [25]:

```
wine_temp = wine[wine['goodquality']==1]#good quality

wine_temp2 = wine[wine['goodquality']==0]#bad quality
```

In [26]:

```
mean1 = pd.Series(wine_temp.mean(), index=['alcohol', 'volatile acidity', 'sulphates', '
density'], name='GoodQuality')
mean2 = pd.Series(wine_temp2.mean(), index=['alcohol', 'volatile acidity', 'sulphates',
'density'], name='BadQuality')
df=pd.concat([mean1, mean2], axis=1)
df.style.highlight_max(color = 'yellow', axis = 1)
```

Out[26]:

	GoodQuality	BadQuality
alcohol	**11.518**	10.251
volatile acidity	0.40553	**0.547022**
sulphates	**0.743456**	0.644754
density	0.99603	**0.996859**

By looking into the details, we can see that good quality wines have higher levels of alcohol on average, have a lower volatile acidity on average, higher levels of sulphates on average, and lower levels of density on average.