In [1]:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```python
df=pd.read_csv('https://raw.githubusercontent.com/dsrscientist/DSData/master/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

In [3]:

```python
df.shape ##Shape of the dataset
```

Out[3]:

```
(1599, 12)
```

In [4]:

```python
df.dtypes
```

Out[4]:

```
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
```

In [5]:

```python
df.isnull().sum() ## checking for null values
```

Out[5]:

```
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 [ ]:

```python

```

In [6]:

```python
## check for descriptive statistics
```

In [7]:

```python
df.describe()
```

Out[7]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	159
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113	0.658149	1

std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0.169507	
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000	
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.550000	
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0.620000	1
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0.730000	1
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.000000	1

In [8]:
```
x=df.drop(columns=['quality'])
print(x)
```

```
      fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0               7.4             0.700         0.00             1.9      0.076
1               7.8             0.880         0.00             2.6      0.098
2               7.8             0.760         0.04             2.3      0.092
3              11.2             0.280         0.56             1.9      0.075
4               7.4             0.700         0.00             1.9      0.076
...             ...               ...          ...             ...        ...
1594            6.2             0.600         0.08             2.0      0.090
1595            5.9             0.550         0.10             2.2      0.062
1596            6.3             0.510         0.13             2.3      0.076
1597            5.9             0.645         0.12             2.0      0.075
1598            6.0             0.310         0.47             3.6      0.067

      free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                    11.0                  34.0  0.99780  3.51       0.56
1                    25.0                  67.0  0.99680  3.20       0.68
2                    15.0                  54.0  0.99700  3.26       0.65
3                    17.0                  60.0  0.99800  3.16       0.58
4                    11.0                  34.0  0.99780  3.51       0.56
...                   ...                   ...      ...   ...        ...
1594                 32.0                  44.0  0.99490  3.45       0.58
1595                 39.0                  51.0  0.99512  3.52       0.76
1596                 29.0                  40.0  0.99574  3.42       0.75
1597                 32.0                  44.0  0.99547  3.57       0.71
1598                 18.0                  42.0  0.99549  3.39       0.66

      alcohol
0         9.4
1         9.8
2         9.8
3         9.8
4         9.4
...       ...
1594     10.5
1595     11.2
1596     11.0
1597     10.2
1598     11.0

[1599 rows x 11 columns]
```

In [9]:
```
y=df['quality']
print(y)
```

```
0       5
1       5
2       5
3       6
4       5
       ..
1594    5
1595    6
1596    6
1597    5
1598    6
Name: quality, Length: 1599, dtype: int64
```

In [10]:
```
##Remove outliers
```

In [11]:
```
from scipy.stats import zscore
```

In [12]:
```
z_score=zscore(df[['fixed acidity','volatile acidity','citric acid','residual sugar','chlorides','free sulfur dio
abs_z_score=np.abs(z_score)
filtered_entry=(abs_z_score<3).all(axis=1)
df=df[filtered_entry]
```

In [13]:
```
df.describe()
```

Out[13]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	
count	1458.000000	1458.000000	1458.000000	1458.000000	1458.000000	1458.000000	1458.000000	1458.000000	1458.000000	1458.000000	145
mean	8.312551	0.524050	0.265281	2.388717	0.081531	15.089849	43.660494	0.996718	3.316152	0.642414	1
std	1.647635	0.169451	0.191271	0.865307	0.021218	9.317669	29.414615	0.001718	0.141052	0.129753	
min	5.000000	0.120000	0.000000	1.200000	0.038000	1.000000	6.000000	0.991500	2.880000	0.330000	
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	21.000000	0.995600	3.220000	0.550000	
50%	7.900000	0.520000	0.250000	2.200000	0.079000	13.000000	36.000000	0.996700	3.315000	0.620000	1
75%	9.200000	0.635000	0.420000	2.600000	0.089000	21.000000	58.000000	0.997800	3.400000	0.720000	1
max	13.500000	1.040000	0.790000	6.700000	0.226000	47.000000	145.000000	1.002200	3.750000	1.160000	1

In [14]:
```
##df['quality']=df['quality'].apply(lambda quality:1 if quality>=6 else 0)
##df['quality']
```

In [15]:
```
df.shape
```

Out[15]:
```
(1458, 12)
```

In [16]:
```
df.head()
```

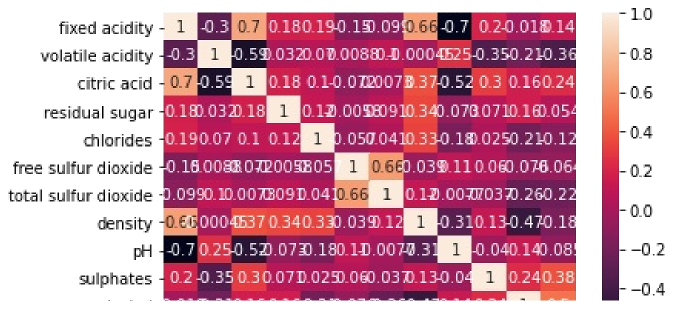
Out[16]:

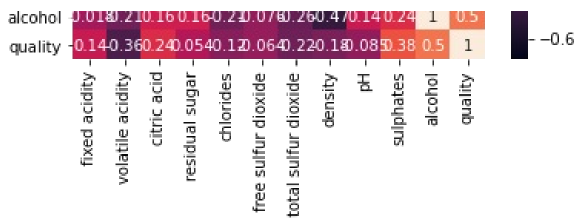
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970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

In [17]:
```
df['quality'].value_counts()
```

Out[17]:
```
5    617
6    586
7    185
4     47
8     16
3      7
Name: quality, dtype: int64
```

In [18]:
```
corr=df.corr()
sns.heatmap(corr,annot=True)
plt.show()
```





In [19]:
```
## there is no accurate information present in the heatmap
## so we opt another method to select best features
```

In [20]:
```
##
```

In [21]:
```
##
```

In [22]:
```
from sklearn.feature_selection import SelectKBest,f_classif
```

In [23]:
```
best_features=SelectKBest(score_func=f_classif,k=7)
fit=best_features.fit(x,y)
df_scores=pd.DataFrame(fit.scores_)
df_columns=pd.DataFrame(x.columns)
```

In [24]:
```
features_score=pd.concat([df_columns,df_scores],axis=1)
features_score.columns=['feature_Name','Score']
print(features_score.nlargest(7,'Score'))
```

```
            feature_Name       Score
10               alcohol  115.854797
1       volatile acidity   60.913993
6   total sulfur dioxide   25.478510
9              sulphates   22.273376
2            citric acid   19.690664
7                density   13.396357
0          fixed acidity    6.283081
```

In [25]:
```
## Model Building
```

In [26]:
```
x=df[['alcohol','volatile acidity','total sulfur dioxide','sulphates','citric acid','density','chlorides',]]
print(x)
```

```
      alcohol  volatile acidity  total sulfur dioxide  sulphates  citric acid  \
0         9.4             0.700                  34.0       0.56         0.00
1         9.8             0.880                  67.0       0.68         0.00
2         9.8             0.760                  54.0       0.65         0.04
3         9.8             0.280                  60.0       0.58         0.56
4         9.4             0.700                  34.0       0.56         0.00
...       ...               ...                   ...        ...          ...
1594     10.5             0.600                  44.0       0.58         0.08
1595     11.2             0.550                  51.0       0.76         0.10
1596     11.0             0.510                  40.0       0.75         0.13
1597     10.2             0.645                  44.0       0.71         0.12
1598     11.0             0.310                  42.0       0.66         0.47

      density  chlorides
0     0.99780      0.076
1     0.99680      0.098
2     0.99700      0.092
3     0.99800      0.075
4     0.99780      0.076
...       ...        ...
1594  0.99490      0.090
1595  0.99512      0.062
1596  0.99574      0.076
1597  0.99547      0.075
1598  0.99549      0.067

[1458 rows x 7 columns]
```

In [27]:
```
x.shape
```
Out[27]:
```
(1458, 7)
```

In [28]:
```
y=df['quality']
print(y)
y.shape
```
```
0       5
1       5
2       5
3       6
4       5
       ..
1594    5
1595    6
1596    6
1597    5
1598    6
Name: quality, Length: 1458, dtype: int64
```
Out[28]:
```
(1458,)
```

In [29]:
```
##
```

In [30]:
```
from sklearn.preprocessing import StandardScaler
scalar=StandardScaler()
X_scalar=scalar.fit_transform(x)
```

In [31]:
```
from sklearn.model_selection import train_test_split
```

In [32]:
```
from sklearn.metrics import accuracy_score,confusion_matrix,roc_curve,roc_auc_score
```

In [33]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=33)
```

In [34]:
```
##
```

In [35]:
```
##y=df['quality']
##print(y)
```

In [36]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [37]:
```
clf=DecisionTreeClassifier()
clf.fit(x_train,y_train)
clf.score(x_train,y_train)
```
Out[37]:
```
1.0
```

In [38]:
```
y_pred=clf.predict(x_test)
```

In [39]:
```
accuracy_score(y_test,y_pred)
```
Out[39]:
```
0.5972602739726027
```

In [40]:
```
confusion_matrix(y_test,y_pred)
```
Out[40]:
```
array([[  1,   7,   1,   0,   0],
       [  4, 106,  34,   5,   0],
       [  5,  41,  87,  19,   2],
       [  0,   5,  18,  24,   1],
       [  0,   0,   5,   0,   0]], dtype=int64)
```

In [41]:
```
from sklearn.svm import SVC
```

In [42]:
```
svc=SVC()
```

In [43]:
```
svc.fit(x_train,y_train)
```

Out[43]:
```
SVC()
```

In [44]:
```
svc.score(x_train,y_train)
```

Out[44]:
```
0.49130832570905764
```

In [45]:
```
from sklearn.ensemble import GradientBoostingClassifier
```

In [46]:
```
gb=GradientBoostingClassifier()
```

In [47]:
```
gb.fit(x_train,y_train)
```

Out[47]:
```
GradientBoostingClassifier()
```

In [48]:
```
gb.score(x_train,y_train)
```

Out[48]:
```
0.8865507776761208
```

In [49]:
```
y_pred=gb.predict(x_test)
```

In [50]:
```
accuracy_score(y_test,y_pred)
```

Out[50]:
```
0.6684931506849315
```

In [51]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [52]:
```
rf=RandomForestClassifier()
rf.fit(x_train,y_train)
```

Out[52]:
```
RandomForestClassifier()
```

In [53]:
```
rf.score(x_train,y_train)
```

Out[53]:
```
1.0
```

In [54]:
```
accuracy_score(y_test,y_pred)
```

Out[54]:
```
0.6684931506849315
```

In [55]:
```
from sklearn.linear_model import LogisticRegression
```

In [56]:
```
lr=LogisticRegression()
```

In [57]:
```
lr.fit(x_train,y_train)
```

```
C:\Users\Rakesh Lodem\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfg
s failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[57]:
```
LogisticRegression()
```

In [58]:
```
lr.score(x_train,y_train)
```

Out[58]:
```
0.5818847209515096
```

In [59]:
```
y_pred=lr.predict(x_test)
```

In [60]:
```
accuracy_score(y_test,y_pred)
```

Out[60]:
```
0.547945205479452
```

In [ ]:
```
#hyperparamter tuning
```

In [ ]:
```

```

In [ ]:
```
gb=GradientBoostingClassifier()
```

In [ ]:
```
parameters={'mini_samples_split':range(4,8,2),'max_depth':[1,3,5,7,9],'learning_rate':[0.01,0.1,1,10,100]}
```

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

In [ ]:
```
cv=GridSearchCV(gb,parameters=parameters,cv=5)
```

In [ ]:
```
cv.fit(x_train,y_train)
```

In [ ]:
```
print(cv.best_params_)
```

In [ ]:
```

```

In [ ]:
```

```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js