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

In [2]:
```
# reading the wine dataset
df = pd.read_csv('https://raw.githubusercontent.com/shrikant-temburwar/Wine-Quali
```

In [3]:
```
# first five rows
df.head()
```

Out[3]:

	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 [4]:
```
# statistical analysis of data
df.describe().T
```

Out[4]:

	count	mean	std	min	25%	50%	75%	max
fixed acidity	1599.0	8.319637	1.741096	4.60000	7.1000	7.90000	9.200000	15.90000
volatile acidity	1599.0	0.527821	0.179060	0.12000	0.3900	0.52000	0.640000	1.58000
citric acid	1599.0	0.270976	0.194801	0.00000	0.0900	0.26000	0.420000	1.00000
residual sugar	1599.0	2.538806	1.409928	0.90000	1.9000	2.20000	2.600000	15.50000
chlorides	1599.0	0.087467	0.047065	0.01200	0.0700	0.07900	0.090000	0.61100
free sulfur dioxide	1599.0	15.874922	10.460157	1.00000	7.0000	14.00000	21.000000	72.00000
total sulfur dioxide	1599.0	46.467792	32.895324	6.00000	22.0000	38.00000	62.000000	289.00000
density	1599.0	0.996747	0.001887	0.99007	0.9956	0.99675	0.997835	1.00369
pH	1599.0	3.311113	0.154386	2.74000	3.2100	3.31000	3.400000	4.01000
sulphates	1599.0	0.658149	0.169507	0.33000	0.5500	0.62000	0.730000	2.00000
alcohol	1599.0	10.422983	1.065668	8.40000	9.5000	10.20000	11.100000	14.90000
quality	1599.0	5.636023	0.807569	3.00000	5.0000	6.00000	6.000000	8.00000

In [5]:
```
#finding the columns of data
df.columns
```

Out[5]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

In [6]:
```
# checking the length of the columns of data
len(df.columns)
```

Out[6]:
```
12
```

In [7]:
```
#checking the null values
df.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
```

In [8]:
```python
#checking the dulicated values in our dataset
df.duplicated().sum()
```

Out[8]: 240

In [9]:
```python
#droping of duplicated values
df = df.drop_duplicates()
```

In [10]:
```python
#after dropping the duplicated values checking the dataset
df.head()
```

Out[10]:

	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
5	7.4	0.66	0.00	1.8	0.075	13.0	40.0	0.9978	3.51	0.56	9.4

In [11]:
```python
df.duplicated().sum()
```

Out[11]: 0

Observations : Here we do not having duplicated values

In [13]:
```python
#checking the unique of quality - dependent variable
df['quality'].unique()
```

Out[13]: array([5, 6, 7, 4, 8, 3], dtype=int64)

In [14]:
```python
#checking the value counts
df['quality'].value_counts()
```

Out[14]:
```
5    577
6    535
7    167
4     53
8     17
3     10
Name: quality, dtype: int64
```

In [15]:
```python
#demo on dataframe creation n checking duplicated

df1 = pd.DataFrame([1,2,3,4,54,3,3,4,5,5,5,3,3,3,3,2,2,2])
```

In [16]:
```python
df1.duplicated().sum()
```

Out[16]: 12

```
#Independent and dependent variable
```

In [17]:
```
x = df.iloc[:,:-1]   #df.dfrop['quality']
y = df.iloc[:,-1]    #df['y']
```

In [18]:
```
x.shape , y.shape
```

Out[18]:
```
((1359, 11), (1359,))
```

In [ ]:
```
''' from sklearn.preprocessing import StandardScaler
scaler = StandartScaler()
scaler
scaler.fit_transform(x_train, y_train)'''  # scaling is not required in Decision
```

In [19]:
```
# splitting the data
#importing the train,test split from sklearn
from sklearn.model_selection import train_test_split, GridSearchCV
```

In [20]:
```
x_train, x_test, y_train, y_test = train_test_split(
...     x, y, test_size=0.33, random_state=42)
```

In [21]:
```
# importing the decision tree classifier from sklearn
from sklearn.tree import DecisionTreeClassifier
```

In [22]:
```
DT = DecisionTreeClassifier()
```

In [23]:
```
DT
```

Out[23]:
```
DecisionTreeClassifier()
```

In [24]:
```
#fittign our data in decisionTree model
DT.fit(x_train,y_train)
```

Out[24]:
```
DecisionTreeClassifier()
```

In [25]:
```
#training dataset accuracy
DT.score(x_train,y_train)
```

Out[25]:
```
1.0
```

In [26]:
```
#prediction of x_test
y_pred = DT.predict(x_test)
```

In [27]:
```
#checking accuracy
from sklearn.metrics import accuracy_score
```

In [28]:
```
accuracy_score(y_test, y_pred) #model accuracy (test accuracy)
```
Out[28]:
```
0.46325167037861914
```

Observations: From DecisionTree model, the accuracy is 47%

Observations: Training accuracy (Bias) is high and test accuracy(variance) is low. Hense this scenario is called Overfitting

In [29]:
```
#model 2: Logistic Regression
from sklearn.linear_model import LogisticRegression
```

In [30]:
```
lr = LogisticRegression()
```

In [31]:
```
lr
```
Out[31]:
```
LogisticRegression()
```

In [32]:
```
#fitting the data in logistic regression
lr.fit(x_train, y_train)
```
Out[32]:
```
LogisticRegression()
```

In [33]:
```
#predicting value using logistic regression
y_predlr = lr.predict(x_test)
```

In [34]:
```
#checking accuracy score of logistic regression
accuracy_score(y_test, y_predlr)
```
Out[34]:
```
0.5902004454342984
```

Observation: From Logtistic Regression we got 59% accuracy

In [35]:
```
#model 3: importing SVC

from sklearn.svm import SVC
```

In [36]:
```
svc = SVC()
```

In [37]:
```
#fitting our data into svc model
svc.fit(x_train,y_train)
```
Out[37]:
```
SVC()
```

In [38]:
```
# predicting the value of x_test in svc model
y_predsvc = svc.predict(x_test)
```

In [39]:
```
accuracy_score(y_test, y_predsvc)
```

Out[39]:
```
0.512249443207127
```

Observations: From SVC model we got 51% of accuracy

In [40]:
```
#creating parameters for grid_search CV
grid_params = {'criterion' : ['gini', 'entropy'],
               'max_depth' : range(2, 32,1),
               'min_samples_leaf' : range(1,10,1),
               'min_samples_split' : range(2,10,1),
               'splitter':['best', 'random']}
```

In [41]:
```
grid_search = GridSearchCV(estimator=DT,
             param_grid=grid_params, cv = 5)
```

In [42]:
```
#fitting our data into gridsearch
grid_search.fit(x_train, y_train)
```

Out[42]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(2, 32),
                         'min_samples_leaf': range(1, 10),
                         'min_samples_split': range(2, 10),
                         'splitter': ['best', 'random']})
```

In [43]:
```
#finding the best params
grid_search.best_params_
```

Out[43]:
```
{'criterion': 'gini',
 'max_depth': 5,
 'min_samples_leaf': 7,
 'min_samples_split': 7,
 'splitter': 'random'}
```

Observations: in Gridsearch CV we found that best parameters are criterion = 'entropy',
    max_depth =4,min_samples_leaf = 4, min_samples_split= 3, splitter= 'random'

In [ ]:
```
#criterion = 'entropy', max_depth =4,min_samples_leaf = 4, min_samples_split= 3,
```

In [44]:
```
#best params with our model(Decision tree)
model_with_best_params = DecisionTreeClassifier(criterion = 'entropy', max_depth
)
```

In [45]:
```
#now fitting the data
model_with_best_params.fit(x_train, y_train)
```

Out[45]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=4, min_samples_leaf=4,
                       min_samples_split=3, splitter='random')
```

In [46]:
```
#predicting the data
y_pred2 = model_with_best_params.predict(x_test)
```

In [47]:
```
#checking the accuracy
accuracy_score(y_test, y_pred2)
```

Out[47]:
```
0.579064587973274
```

Observation: Here we got 57 % of accuracy

In [ ]: