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')
%matplotlib inline
```

In [2]:
```
#reading the dataset from github
df = pd.read_csv('https://raw.githubusercontent.com/amberkakkar01/Prediction-of-W
```

In [3]:
```
#dataframe head..giving first five rows of data
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]:
```
#finding the columns
df.columns
```

Out[4]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

```
Observations: Here we found 12 columns(Features), where 12 are independent
features and 1 is dependent feature
```

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

```
Observations: We found 12 features data type is float and 1 dependent
features'is integer
```

In [6]:
```
#checking null values of data
df.isnull().sum()
```

Out[6]:
```
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
```

```
Observations: No null values found here
```

In [7]:
```
#let us check the how many unique values are there in dependent feature
df.quality.unique()
```

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

In [8]:
```
#checking the values and counts of quality feature
df.quality.value_counts()
```

Out[8]:
```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

In [9]:
```
#checking index index of quality feature
df.quality.value_counts().index
```

Out[9]:
```
Int64Index([5, 6, 7, 4, 8, 3], dtype='int64')
```

In [10]:
```
#checking index values of quality feature
df.quality.value_counts().values
```

Out[10]:
```
array([681, 638, 199,  53,  18,  10], dtype=int64)
```

In [11]:
```
#performing statistical analysis
df.describe(include='all').T
```

Out[11]:

	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 [12]:
```
x = df.iloc[:,:-1] #Seperating independent features from given dataset
```

In [13]:
```
y = df.iloc[:,-1] #Seperating independent features from given dataset
```

In [14]:
```
x
```

Out[14]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9
...	...	...	...	...	...	...	...	...	...	...	
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11

1599 rows × 11 columns

In [15]:
```
y
```

Out[15]:
```
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 [16]:
```
#Performing standardization from sklearn
from sklearn.preprocessing import StandardScaler
```

In [17]:
```
#splitting the data into train and test
from sklearn.model_selection import train_test_split
```

In [18]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_
```

In [19]:
```python
x_train.head()
```

Out[19]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
548	12.4	0.350	0.49	2.6	0.079	27.0	69.0	0.99940	3.12	0.75	10
355	6.7	0.750	0.01	2.4	0.078	17.0	32.0	0.99550	3.55	0.61	12
1296	6.6	0.630	0.00	4.3	0.093	51.0	77.5	0.99558	3.20	0.45	9
209	11.0	0.300	0.58	2.1	0.054	7.0	19.0	0.99800	3.31	0.88	10
140	8.4	0.745	0.11	1.9	0.090	16.0	63.0	0.99650	3.19	0.82	9

In [20]:
```python
x_test.head()
```

Out[20]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
803	7.7	0.56	0.08	2.50	0.114	14.0	46.0	0.9971	3.24	0.66	9
124	7.8	0.50	0.17	1.60	0.082	21.0	102.0	0.9960	3.39	0.48	9
350	10.7	0.67	0.22	2.70	0.107	17.0	34.0	1.0004	3.28	0.98	9
682	8.5	0.46	0.31	2.25	0.078	32.0	58.0	0.9980	3.33	0.54	9
1326	6.7	0.46	0.24	1.70	0.077	18.0	34.0	0.9948	3.39	0.60	10

In [21]:
```python
scaler = StandardScaler()
```

In [22]:
```python
scaler
```

Out[22]:
```
StandardScaler()
```

In [23]:
```python
#fitting the train data
print(scaler.fit(x_train))
```

```
StandardScaler()
```

In [24]:
```python
#finding the mean
print(scaler.mean_)
```

```
[ 8.30345472  0.53246499  0.26933707  2.54691877  0.08772736 15.91223156
 46.76330532  0.99677933  3.31453782  0.65881419 10.41521942]
```

In [25]:
```
#transform the training data
scaler.transform(x_train)
```

Out[25]:
```
array([[ 2.40069523, -1.03103722,  1.12742595, ..., -1.26096312,
         0.52726134, -0.01431863],
       [-0.93967131,  1.22920403, -1.32502245, ...,  1.52622836,
        -0.28225704,  2.24363201],
       [-0.99827424,  0.55113165, -1.37611513, ..., -0.74241587,
        -1.20742091, -0.86105011],
       ...,
       [-0.6466567 ,  0.49462562, -1.06955908, ...,  1.26695473,
        -0.68701624, -0.86105011],
       [-0.23643625, -1.87862768,  0.4121285 , ...,  0.03540501,
         0.81637505,  1.39690052],
       [-1.46709761, -1.3700734 , -0.04770558, ...,  0.48913386,
        -0.68701624,  2.90220094]])
```

In [26]:
```
#another form of fit_transform
x_train_tf = scaler.fit_transform(x_train)
```

In [27]:
```
x_train_tf
```

Out[27]:
```
array([[ 2.40069523, -1.03103722,  1.12742595, ..., -1.26096312,
         0.52726134, -0.01431863],
       [-0.93967131,  1.22920403, -1.32502245, ...,  1.52622836,
        -0.28225704,  2.24363201],
       [-0.99827424,  0.55113165, -1.37611513, ..., -0.74241587,
        -1.20742091, -0.86105011],
       ...,
       [-0.6466567 ,  0.49462562, -1.06955908, ...,  1.26695473,
        -0.68701624, -0.86105011],
       [-0.23643625, -1.87862768,  0.4121285 , ...,  0.03540501,
         0.81637505,  1.39690052],
       [-1.46709761, -1.3700734 , -0.04770558, ...,  0.48913386,
        -0.68701624,  2.90220094]])
```

Model building

importing SVC model from sklearn

In [28]:
```
from sklearn.svm import SVC
```

In [29]:
```
model = SVC()
```

In [30]:
```
#fitting our data into this model
model.fit(x_train_tf, y_train )
```

Out[30]:
```
SVC()
```

In [31]:
```
model.score(x_train_tf, y_train) # checking training accuracy
```

Out[31]:
```
0.6778711484593838
```

```
Observation: Here we got 67% accuracy for train data
```

In [33]:
```
#need to transform the test data
x_test_tf = scaler.transform(x_test)
```

In [34]:
```
x_test_tf
```

Out[34]:
```
array([[-3.53642095e-01,  1.55589436e-01, -9.67373729e-01, ...,
        -4.83142240e-01,  6.85666499e-03, -7.66968836e-01],
       [-2.95039173e-01, -1.83446751e-01, -5.07539654e-01, ...,
         4.89133857e-01, -1.03395269e+00, -8.61050113e-01],
       [ 1.40444556e+00,  7.77155778e-01, -2.52076279e-01, ...,
        -2.23868614e-01,  1.85718440e+00, -4.84725007e-01],
       ...,
       [-2.02456406e-03, -1.25706134e+00,  6.16499196e-01, ...,
        -2.94133945e-02,  6.42906824e-01,  1.96138818e+00],
       [-6.06274859e-02,  4.50655383e+00, -1.37611513e+00, ...,
         1.39659155e+00, -9.76129945e-01,  4.56087756e-01],
       [ 4.66798811e-01,  7.20649747e-01, -6.09725004e-01, ...,
        -2.23868614e-01, -6.87016236e-01, -7.66968836e-01]])
```

In [35]:
```
y_predict = model.predict(x_test_tf)
```

In [36]:
```
y_test
```

Out[36]:
```
803     6
124     5
350     6
682     5
1326    6
       ..
813     4
377     7
898     7
126     5
819     5
Name: quality, Length: 528, dtype: int64
```

In [37]:
```
y_predict
```

Out[37]:
```
array([5, 5, 6, 5, 6, 5, 5, 5, 6, 6, 6, 5, 6, 5, 5, 7, 5, 6, 7, 5, 5, 5,
       6, 6, 5, 5, 6, 5, 5, 6, 5, 5, 6, 5, 6, 5, 6, 6, 5, 6, 5, 5, 6, 5,
       6, 6, 6, 6, 5, 6, 5, 5, 6, 7, 5, 5, 6, 5, 6, 5, 6, 6, 5, 5, 7, 5,
       6, 5, 7, 5, 6, 5, 6, 6, 6, 5, 7, 5, 6, 7, 5, 7, 5, 5, 6, 6, 5, 6,
       6, 5, 6, 5, 5, 6, 5, 6, 5, 6, 5, 5, 5, 5, 6, 6, 6, 6, 6, 5, 6, 5,
       6, 5, 6, 5, 6, 6, 6, 5, 5, 6, 6, 6, 6, 5, 5, 5, 6, 6, 5, 6, 6, 5,
       5, 6, 6, 5, 5, 5, 5, 6, 6, 6, 6, 5, 6, 5, 6, 5, 6, 5, 6, 6, 5, 6,
       6, 6, 5, 6, 5, 6, 7, 6, 6, 5, 5, 6, 5, 5, 5, 5, 5, 5, 6, 5, 7, 6,
       6, 5, 5, 5, 5, 7, 5, 7, 5, 6, 6, 6, 7, 5, 6, 6, 5, 6, 6, 5, 5, 5,
       6, 6, 5, 5, 5, 5, 7, 6, 5, 5, 6, 6, 7, 5, 6, 6, 6, 6, 6, 5, 6, 5,
       5, 6, 6, 6, 5, 5, 5, 7, 5, 5, 5, 5, 6, 6, 5, 6, 5, 6, 6, 5, 5, 5,
       6, 6, 5, 6, 6, 5, 6, 5, 6, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 5, 7,
       6, 7, 6, 5, 6, 6, 5, 6, 5, 5, 5, 5, 6, 6, 6, 5, 7, 5, 5, 5, 5, 6,
       5, 6, 5, 6, 5, 7, 6, 5, 5, 6, 5, 6, 6, 7, 5, 5, 6, 5, 5, 5, 6, 6,
       6, 7, 5, 5, 6, 5, 5, 6, 5, 5, 6, 5, 6, 5, 6, 5, 5, 5, 6, 5, 5, 6,
       6, 7, 5, 5, 6, 6, 6, 6, 5, 5, 6, 7, 5, 5, 6, 5, 6, 5, 6, 6, 6, 6,
       5, 5, 6, 6, 5, 5, 5, 5, 5, 5, 5, 6, 5, 6, 6, 5, 5, 5, 5, 5, 6, 6,
       5, 6, 5, 6, 5, 5, 5, 6, 6, 5, 6, 6, 6, 5, 5, 6, 5, 5, 5, 6, 6, 6,
       7, 6, 5, 6, 5, 5, 6, 5, 5, 6, 7, 6, 5, 5, 6, 7, 6, 6, 6, 6, 5, 7,
       5, 6, 6, 5, 5, 5, 6, 6, 5, 5, 6, 5, 7, 5, 5, 5, 6, 5, 5, 5, 5, 6,
       6, 6, 6, 5, 5, 5, 5, 6, 6, 5, 6, 6, 5, 5, 5, 6, 7, 6, 6, 5, 5, 5,
       5, 5, 6, 5, 5, 5, 5, 6, 7, 6, 6, 6, 5, 6, 6, 6, 6, 5, 6, 6, 6, 6,
       5, 6, 6, 6, 5, 5, 6, 6, 5, 5, 6, 5, 6, 5, 5, 5, 5, 5, 5, 5, 5, 5,
       6, 6, 6, 6, 6, 6, 5, 5, 5, 7, 6, 6, 6, 5, 5, 5, 6, 6, 7, 7, 5, 5],
      dtype=int64)
```

In [38]:
```
# checking accuracy
from sklearn.metrics import accuracy_score
```

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

Out[39]:
```
0.5984848484848485
```

In [40]:
```
# checking the accuracy Using logistic regression model
from sklearn.linear_model import LogisticRegression
```

In [41]:
```
model2 = LogisticRegression()
```

In [42]:
```
model2
```

Out[42]:
```
LogisticRegression()
```

In [43]:
```
model2.fit(x_train_tf, y_train)
```

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

In [44]:
```
#predicting the test data in logistic regression model
y_predict2 = model2.predict(x_test_tf)
```

In [45]:
```
y_predict2
```

Out[45]:
```
array([5, 5, 6, 5, 6, 5, 5, 5, 6, 6, 6, 5, 6, 5, 5, 7, 5, 5, 7, 5, 5, 5,
       6, 6, 5, 5, 6, 5, 5, 6, 5, 5, 6, 5, 6, 5, 6, 6, 5, 6, 5, 5, 6, 5,
       6, 6, 6, 5, 5, 6, 5, 5, 6, 6, 5, 5, 6, 5, 6, 5, 5, 6, 5, 5, 7, 5,
       7, 5, 6, 5, 7, 5, 6, 6, 6, 5, 7, 6, 6, 7, 5, 7, 5, 6, 6, 6, 5, 6,
       6, 5, 6, 5, 6, 6, 5, 6, 5, 6, 5, 6, 5, 5, 6, 6, 6, 6, 5, 5, 6, 5,
       7, 5, 6, 5, 6, 6, 6, 5, 5, 6, 6, 5, 6, 5, 5, 5, 6, 6, 6, 6, 6, 5,
       5, 6, 6, 5, 5, 5, 5, 6, 6, 6, 6, 6, 6, 5, 6, 5, 6, 5, 6, 6, 5, 6,
       6, 6, 5, 6, 5, 6, 6, 6, 6, 5, 5, 6, 5, 5, 5, 5, 5, 5, 6, 5, 5, 6,
       6, 5, 5, 5, 5, 6, 5, 7, 5, 6, 6, 6, 7, 5, 6, 6, 6, 6, 6, 5, 5, 5,
       5, 6, 5, 5, 5, 5, 7, 6, 5, 5, 6, 6, 6, 5, 6, 6, 7, 6, 5, 5, 6, 5,
       5, 6, 6, 6, 5, 5, 5, 7, 5, 5, 5, 5, 6, 6, 6, 6, 5, 6, 5, 5, 5, 5,
       6, 6, 5, 5, 6, 5, 7, 5, 5, 6, 5, 5, 4, 5, 6, 6, 6, 7, 6, 6, 5, 7,
       6, 6, 5, 5, 6, 6, 5, 6, 5, 5, 5, 5, 6, 6, 6, 5, 7, 5, 5, 5, 5, 6,
       5, 6, 5, 6, 5, 7, 5, 5, 5, 6, 5, 6, 6, 7, 5, 5, 6, 5, 5, 5, 6, 6,
       6, 7, 6, 5, 5, 5, 5, 6, 5, 5, 6, 5, 6, 6, 6, 5, 5, 5, 6, 6, 5, 6,
       6, 7, 5, 5, 5, 6, 6, 7, 5, 5, 6, 7, 6, 5, 6, 5, 6, 5, 6, 6, 5, 7,
       5, 5, 6, 6, 5, 5, 5, 6, 6, 5, 6, 6, 5, 6, 6, 5, 5, 5, 5, 5, 6, 6,
       5, 6, 5, 6, 5, 5, 5, 6, 7, 5, 6, 6, 6, 5, 5, 6, 5, 6, 5, 5, 6, 6,
       6, 6, 5, 6, 5, 5, 6, 5, 5, 6, 7, 6, 6, 5, 5, 6, 6, 6, 6, 6, 5, 7,
       5, 6, 6, 5, 6, 5, 6, 6, 5, 5, 6, 5, 7, 5, 5, 5, 7, 5, 5, 5, 5, 6,
       6, 6, 7, 5, 5, 5, 5, 6, 6, 5, 6, 6, 6, 5, 5, 6, 7, 5, 6, 5, 5, 5,
       5, 5, 6, 5, 5, 5, 6, 7, 7, 6, 6, 6, 5, 5, 6, 6, 6, 6, 6, 6, 5, 6,
       5, 6, 6, 6, 5, 5, 6, 6, 5, 5, 6, 5, 6, 5, 5, 5, 5, 5, 5, 5, 6, 5,
       5, 6, 6, 6, 6, 6, 5, 3, 5, 7, 7, 5, 5, 5, 5, 5, 7, 6, 7, 7, 3, 5],
      dtype=int64)
```

In [46]:
```
accuracy_score(y_test,y_predict2)
```

Out[46]:
```
0.571969696969697
```

Observations: We found that, we got only 57% accuracy using logistic Regression model

In [ ]:

In [ ]: