In [69]:

```
import numpy, pandas, scipy
#Rescaling the data
from sklearn.preprocessing import MinMaxScaler
df=pandas.read_csv("C:/Users/ABHISHEK/Desktop/GITAM ML/GITAM ML LAB/redwinequality.csv")
df.shape
```

Out[69]:

```
(1599, 12)
```

In [45]:

```
df.head()
```

Out[45]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9

In [70]:

```
df.describe()
```

Out[70]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total su dio
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000

In [73]:

```
#Seperating data into input and output components
x=df.iloc[:,0:11]
y=df.iloc[:,11:]
df.describe()
```

Out[73]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total su dio
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000

In [74]:

```
x.head(2)
```

Out[74]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
0	7.4	0.70	0.0	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9
1	7.8	0.88	0.0	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9

In [80]:

```
#if no argument then prints all y
y.head(3)
```

Out[80]:

	quality
0	5
1	5
2	5

In [81]:

```
#defining the scaler
scaler=MinMaxScaler(feature_range=(0,1))
print(scaler)
print(type(scaler))
```

```
MinMaxScaler(copy=True, feature_range=(0, 1))
<class 'sklearn.preprocessing.data.MinMaxScaler'>
```

In [82]:

```
rescaledX=scaler.fit_transform(x)
numpy.set_printoptions(precision=2) #Setting precision for the output
print(rescaledX[0:5,:])
```

```
[[0.25 0.4  0.   0.07 0.11 0.14 0.1  0.57 0.61 0.14 0.15]
 [0.28 0.52 0.   0.12 0.14 0.34 0.22 0.49 0.36 0.21 0.22]
 [0.28 0.44 0.04 0.1  0.13 0.2  0.17 0.51 0.41 0.19 0.22]
 [0.58 0.11 0.56 0.07 0.11 0.23 0.19 0.58 0.33 0.15 0.22]
 [0.25 0.4  0.   0.07 0.11 0.14 0.1  0.57 0.61 0.14 0.15]]
```

In [83]:

```
rescaledX.max()
```

Out[83]:

```
1.0000000000000002
```

In [84]:

```
from sklearn.preprocessing import StandardScaler
#code for standardizing the data
scaler=StandardScaler().fit(x)
print(scaler)
print(type(scaler))
```

```
StandardScaler(copy=True, with_mean=True, with_std=True)
<class 'sklearn.preprocessing.data.StandardScaler'>
```

In [85]:

```
rescaledX=scaler.transform(x)
print(rescaledX[0:5,:])
```

```
[[-0.53  0.96 -1.39 -0.45 -0.24 -0.47 -0.38  0.56  1.29 -0.58 -0.96]
 [-0.3   1.97 -1.39  0.04  0.22  0.87  0.62  0.03 -0.72  0.13 -0.58]
 [-0.3   1.3  -1.19 -0.17  0.1  -0.08  0.23  0.13 -0.33 -0.05 -0.58]
 [ 1.65 -1.38  1.48 -0.45 -0.26  0.11  0.41  0.66 -0.98 -0.46 -0.58]
 [-0.53  0.96 -1.39 -0.45 -0.24 -0.47 -0.38  0.56  1.29 -0.58 -0.96]]
```

In [86]:

```
rescaledX.mean()
```

Out[86]:

```
2.5450106517076758e-15
```

In [87]:

```
from sklearn.preprocessing import Normalizer
#code for normalizing the data
scaler=Normalizer().fit(x)
normalizedX=scaler.transform(x)
print(normalizedX[0:5,:])
```

```
[[1.95e-01 1.85e-02 0.00e+00 5.01e-02 2.00e-03 2.90e-01 8.97e-01 2.63e-02
  9.26e-02 1.48e-02 2.48e-01]
 [1.07e-01 1.21e-02 0.00e+00 3.57e-02 1.35e-03 3.44e-01 9.21e-01 1.37e-02
  4.40e-02 9.35e-03 1.35e-01]
 [1.35e-01 1.32e-02 6.95e-04 3.99e-02 1.60e-03 2.60e-01 9.38e-01 1.73e-02
  5.66e-02 1.13e-02 1.70e-01]
 [1.74e-01 4.36e-03 8.72e-03 2.96e-02 1.17e-03 2.65e-01 9.34e-01 1.55e-02
  4.92e-02 9.03e-03 1.53e-01]
 [1.95e-01 1.85e-02 0.00e+00 5.01e-02 2.00e-03 2.90e-01 8.97e-01 2.63e-02
  9.26e-02 1.48e-02 2.48e-01]]
```

In [88]:

```
row1 = [1.95152519e-01, 1.84603734e-02, 0.00000000e+00, 5.01067279e-02,
2.00426911e-03, 2.90091582e-01, 8.96646709e-01, 2.63139437e-02,
        9.25655867e-02, 1.47682987e-02, 2.47896443e-01]
res = sum(map(lambda i : i * i, row1))
res
```

Out[88]:

```
0.9999999996141229
```

In [91]:

```
from sklearn.preprocessing import Binarizer
#code for binarizing the data
binarizer=Binarizer(threshold=0.5).fit(x)
binaryX=binarizer.transform(x)
print(binaryX[0:5,:])
```

```
[[1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1.]
 [1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1.]
 [1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1.]
 [1. 0. 1. 1. 0. 1. 1. 1. 1. 1. 1.]
 [1. 1. 0. 1. 0. 1. 1. 1. 1. 1. 1.]]
```

In [92]:

```
print(binaryX.min())
print(binaryX.max())
```

```
0.0
1.0
```

In [94]:

```
print(x.mean(axis=0))
```

```
fixed acidity            8.319637
volatile acidity         0.527821
citric acid              0.270976
residual sugar           2.538806
chlorides                0.087467
free sulfur dioxide     15.874922
total sulfur dioxide    46.467792
density                  0.996747
pH                       3.311113
sulphates                0.658149
alcohol                 10.422983
dtype: float64
```

In [97]:

```
print(x.std(axis=0))
```

```
fixed acidity            1.741096
volatile acidity         0.179060
citric acid              0.194801
residual sugar           1.409928
chlorides                0.047065
free sulfur dioxide     10.460157
total sulfur dioxide    32.895324
density                  0.001887
pH                       0.154386
sulphates                0.169507
alcohol                  1.065668
dtype: float64
```

In [98]:

```
from sklearn.preprocessing import scale
#code for mean removal
data_standardized=scale(x)
```

In [100]:

```
print(data_standardized.mean(axis=0))
print(data_standardized.std(axis=0))
```

```
[ 3.55e-16  1.73e-16 -8.89e-17 -1.24e-16  3.91e-16 -6.22e-17  4.44e-17
  2.36e-14  2.86e-15  6.75e-16  7.11e-17]
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
```

In [101]:

```
# Method1: One-hot encoding directly from pandas using "pandas.get_dummies"
ids = [1, 2, 3, 4, 5, 6, 7]
place = ['Bangladesh','France','Germany','India','Nepal','Spain','Srilanka']
df = pandas.DataFrame(list(zip(ids, place)),
 columns=['Ids', 'Place'])
print(df)
print()
y = pandas.get_dummies(df.Place, prefix='P')
print(y)
```

```
   Ids       Place
0    1  Bangladesh
1    2      France
2    3     Germany
3    4       India
4    5       Nepal
5    6       Spain
6    7    Srilanka

   P_Bangladesh  P_France  P_Germany  P_India  P_Nepal  P_Spain  P_Srilanka
0             1         0          0        0        0        0           0
1             0         1          0        0        0        0           0
2             0         0          1        0        0        0           0
3             0         0          0        1        0        0           0
4             0         0          0        0        1        0           0
5             0         0          0        0        0        1           0
6             0         0          0        0        0        0           1
```

In [108]:

```
# Method2: One-hot encoding using "LabelBinarizer".
from sklearn.preprocessing import LabelBinarizer
ids = [11, 22, 33, 44, 55, 66, 77]
countries = ['France', 'Spain', 'Germany', 'France', 'Spain']
df = pandas.DataFrame(list(zip(ids, countries)),
 columns=['Ids', 'Countries'])
y = LabelBinarizer().fit_transform(df.Countries)
print(y)
```

```
[[1 0 0]
 [0 0 1]
 [0 1 0]
 [1 0 0]
 [0 0 1]]
```

In [110]:

```
# Method3: One-hot encoding using "OneHotEncoder".
from sklearn.preprocessing import OneHotEncoder
x = [[11, "France"], [22, "Spain"], [33, "Germany"], [44, "France"], [55, "Spain"]]
# x = [[1,'Bangladesh'],[2,'France'],[3,'Germany']]
y = OneHotEncoder().fit_transform(x).toarray()
print(y)
```

```
[[1. 0. 0. 0. 0. 1. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 1.]
 [0. 0. 1. 0. 0. 0. 1. 0.]
 [0. 0. 0. 1. 0. 1. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 1.]]
```

In [116]:

```
#Label Encoding
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
input_classes = ['suzuki', 'ford', 'suzuki', 'toyota', 'ford', 'bmw']
label_encoder.fit(input_classes)
print("\nClass mapping: ")
for i, item in enumerate(label_encoder.classes_):
 print(item,'-->', i)
```

```

Class mapping:
bmw --> 0
ford --> 1
suzuki --> 2
toyota --> 3
```

In [119]:

```
# Encoded lables
labels = ['toyota', 'ford','bmw','suzuki','bmw']
encoded_labels = label_encoder.transform(labels)
print("\nLabels =", labels)
print("Encoded labels =", list(encoded_labels))
```

```

Labels = ['toyota', 'ford', 'bmw', 'suzuki', 'bmw']
Encoded labels = [3, 1, 0, 2, 0]
```

In [122]:

```
# Decoding the lables
encoded_labels = [3, 2, 0, 2, 1]
decoded_labels = label_encoder.inverse_transform(encoded_labels)
print("\nEncoded labels =", encoded_labels)
print("Decoded labels =", list(decoded_labels))
```

```

Encoded labels = [3, 2, 0, 2, 1]
Decoded labels = ['toyota', 'suzuki', 'bmw', 'suzuki', 'ford']
```

In [ ]: