In [158…

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import skew
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
```

In [159…

```
#dataset ---> ds

ds=pd.read_csv('QualityPrediction.csv')
ds
```

Out[159…

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoho
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4
...	...	...	...	...	...	...	...	...	...	...	..
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0

1599 rows × 12 columns

ML model

Logistic Regression

In [160…

```
ds.describe()
#ds.isna().sum()
```

Out[160…

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide
count	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

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000

a) Assigning a binary type dependent variable in place of quality

In [161…

```
grade = [] #Declaring a new list
for i in ds['quality']:
    if i >= 7:
        i = 1
        grade.append(i)
    else:
        i = 0
        grade.append(i)
ds['grade'] = grade
ds.drop('quality', axis = 1, inplace = True)
```

In [162…

```
ds
```

Out[162…

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoho
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4
...	...	...	...	...	...	...	...	...	...	...	..
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0

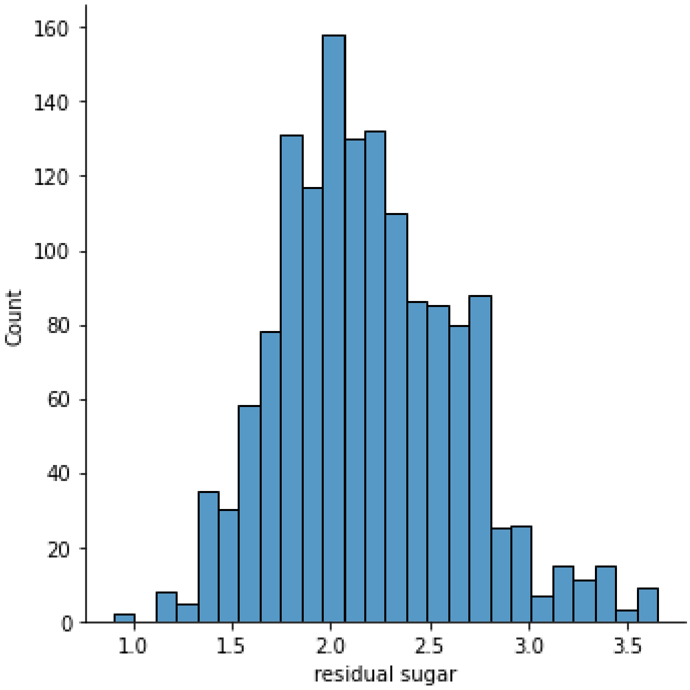
1599 rows × 12 columns

b) Fixing Outliers

In [163…

```
ds.drop(ds[ds['residual sugar']>3.65].index,axis=0,inplace=True)
#ds['residual sugar'].describe()
sns.displot(ds['residual sugar'])
```

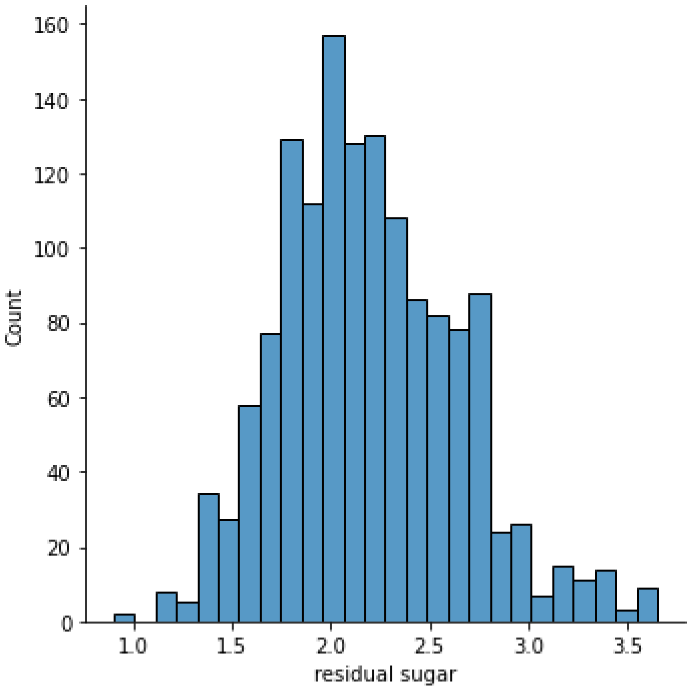
Out[163… <seaborn.axisgrid.FacetGrid at 0x1e3f9454400>



In [164…

```
ds.drop(ds[ds['free sulfur dioxide']>40.875].index,axis=0,inplace=True)
#ds['free sulfur dioxide'].describe()
sns.displot(ds['residual sugar'])
```

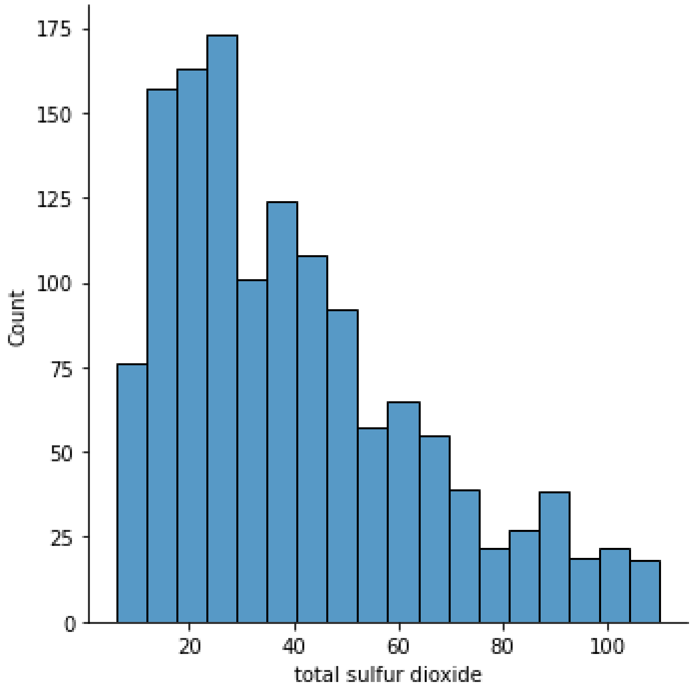
Out[164… <seaborn.axisgrid.FacetGrid at 0x1e3f95c3c70>



In [165…

```
ds.drop(ds[ds['total sulfur dioxide']>110].index,axis=0,inplace=True)
#ds['total sulfur dioxide'].describe()
sns.displot(ds['total sulfur dioxide'])
```

Out[165… <seaborn.axisgrid.FacetGrid at 0x1e3f9b11610>

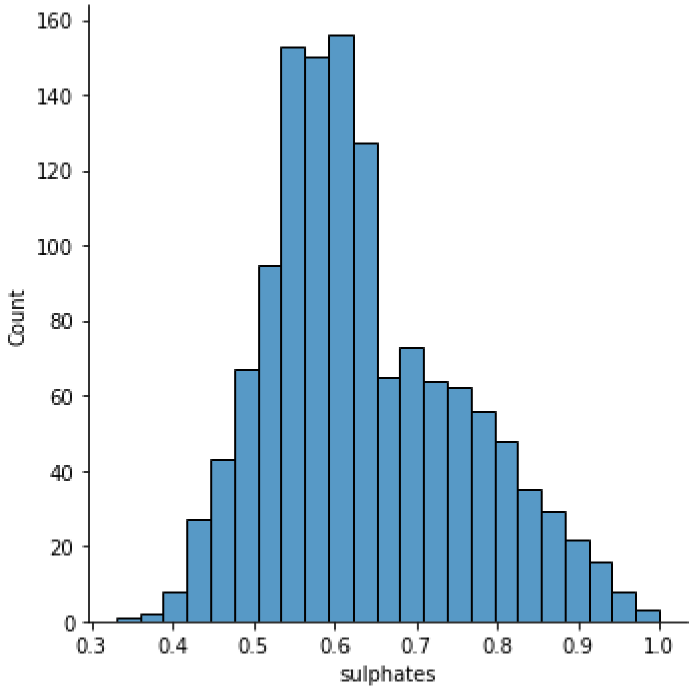


In [166…

```
ds.drop(ds[ds['sulphates']>1].index,axis=0,inplace=True)
#ds['sulphates'].describe()
sns.displot(ds['sulphates'])
```

Out[166…

```
<seaborn.axisgrid.FacetGrid at 0x1e3f96897f0>
```

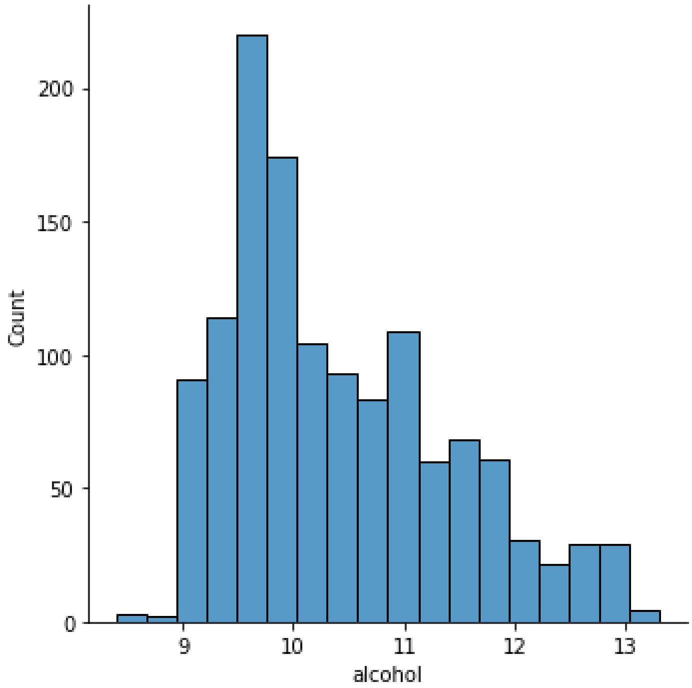


In [186…

```
ds.drop(ds[ds['alcohol']>13.35].index,axis=0,inplace=True)
#ds['alcohol'].describe()
sns.displot(ds['alcohol'])
```

Out[186…

```
<seaborn.axisgrid.FacetGrid at 0x1e3f94ad820>
```



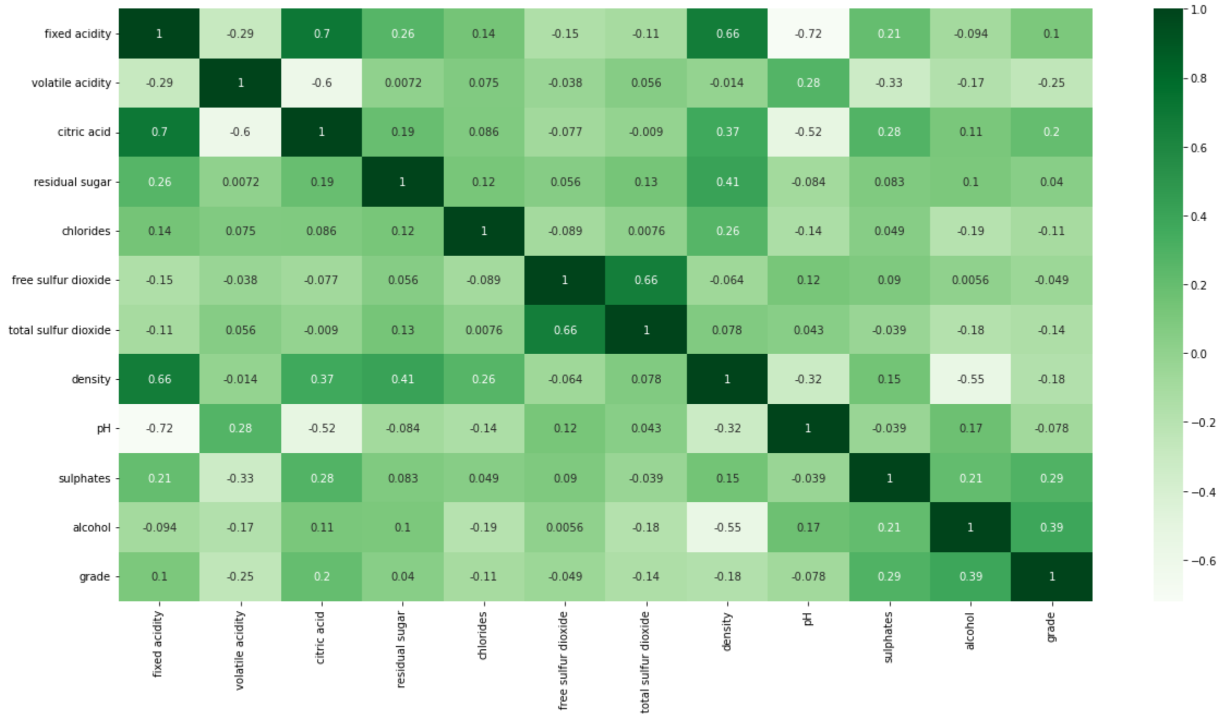
c) Correlation between variables

In [168…

```
plt.figure(figsize=[20,10],facecolor='white')
sns.heatmap(ds.corr(),annot=True, cmap='Greens')
```

Out[168…

```
<AxesSubplot:>
```



In [169…

```
#Checking for multi-collinearity

for a in range(len(ds.corr().columns)):
    for b in range(a):
        if abs(ds.corr().iloc[a,b]) > 0.75:
            A = ds.corr().columns[a]
            print(A)
else:
    print('No multi-collinearity')
```

```
No multi-collinearity
```

d) Train-Test split

In [177…
```
#Independent variables
x=ds.iloc[:,0:-1].values

#Dependent variable
y=ds.iloc[:,-1:].values.ravel()
```

In [178…
```
scaler = StandardScaler()
scaler.fit(x)
scaled_x = scaler.transform(x)

x_train, x_test, y_train, y_test = train_test_split(scaled_x, y , test_size = 0.2, r
```

In [179…
```
logreg = LogisticRegression()
logreg.fit(x_train,y_train)
```

Out[179…
```
LogisticRegression()
```

In [180…
```
y_pred=log_reg.predict(x_test)
y_pred
```

Out[180…
```
array([0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0,
       0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0,
       0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1,
       0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0,
       1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0], dtype=int64)
```

In [184…
```
y_test
```

Out[184…
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0], dtype=int64)
```

e) Evaluation using Visualization

In [187…
```
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(y_test,y_pred)
conf_matrix
```

Out[187…

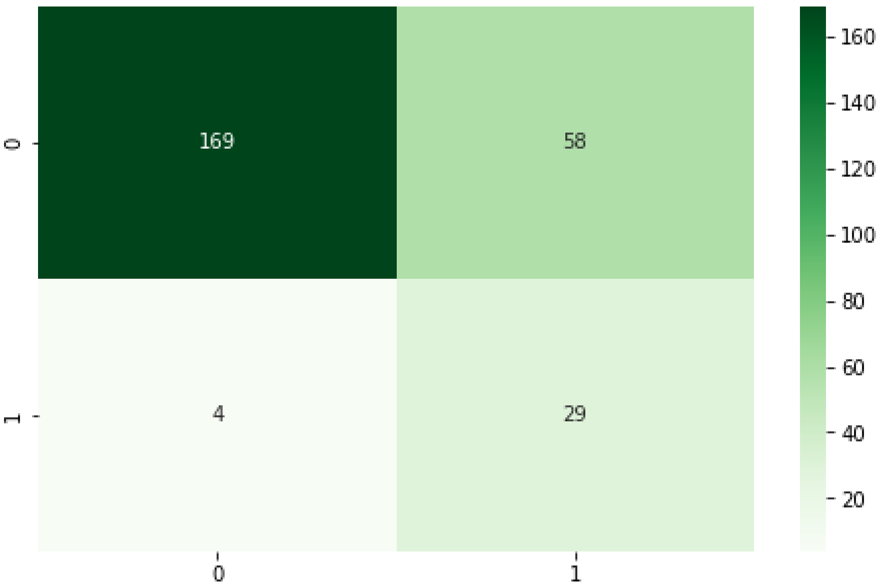
```
array([[169,  58],
       [  4,  29]], dtype=int64)
```

In [196…

```
import seaborn as sns
fig, ax = plt.subplots(figsize=(8,5))
sns.heatmap(conf_matrix, annot = True, cmap='Greens', fmt='g')
```

Out[196…

<AxesSubplot:>



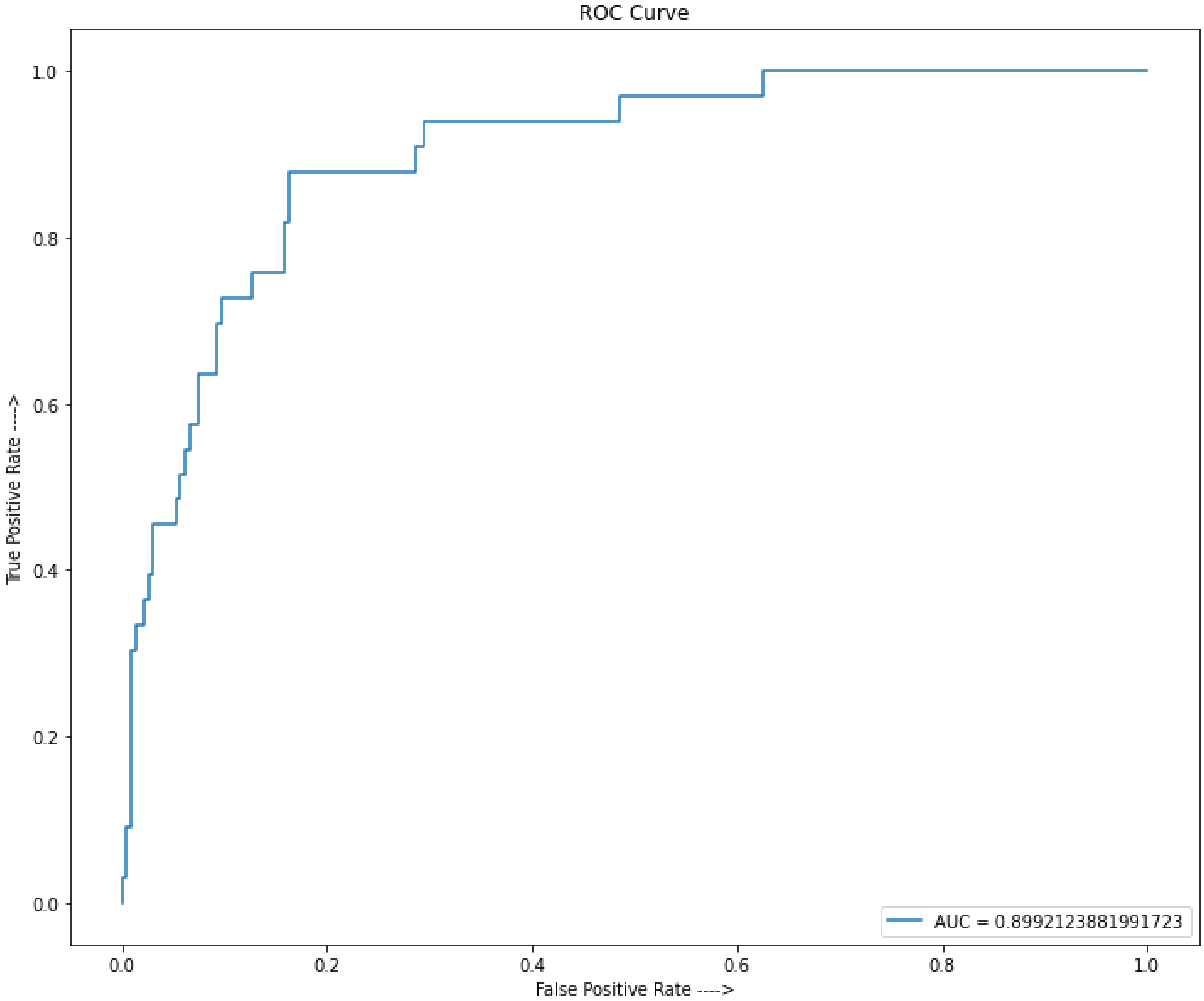
In [197…

```
print("Accuracy: ", metrics.accuracy_score(y_test,y_pred))
print("Precision: ", metrics.precision_score(y_test,y_pred))
print("Recall: ", metrics.recall_score(y_test,y_pred))
```

```
Accuracy:  0.7615384615384615
Precision:  0.3333333333333333
Recall:  0.8787878787878788
```

In [205…

```
y_pred_proba = log_reg.predict_proba(x_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test, y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)
plt.figure(figsize=(12,10))
plt.plot(fpr,tpr,label="AUC = "+str(auc))
plt.legend(loc=4)
plt.title("ROC Curve")
plt.xlabel("False Positive Rate ---->")
plt.ylabel("True Positive Rate ---->")
plt.show()
```


ROC Curve
1.0
0.8
0.6
0.4
0.2
0.0
True Positive Rate ---->
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate ---->
AUC = 0.8992123881991723