In [23]:
```
#import libraries
import pandas as pd
from sklearn.datasets import make_classification
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.metrics import accuracy_score
from sklearn import preprocessing
from sklearn import pipeline
import warnings
warnings.filterwarnings('ignore')
```

In [24]:
```
#import libraries
#import csv_dataset
#check for missing values
#declare axes
#split
#create mode and fit
#test
#accuracy
#optimization
```

In [25]:
```
df = pd.read_csv("C:\\Users\\JOHNSON\\Downloads\\my notes\\linear\\assignment\\
df[:5]
```

Out[25]:

	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 [26]:
```
df.isnull().sum()
```

Out[26]:
```
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 [27]:
```
x=df.drop(['quality'],axis=1)
x
```

Out[27]:

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

1599 rows × 11 columns

In [28]:
```
y=df['quality']
y
```

Out[28]:
```
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 [29]:
```
x_train, x_test, y_train, y_test= train_test_split(x,y, test_size=0.2, random_s
scaler = preprocessing.StandardScaler().fit(x_train)

x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled=scaler.transform(x_test)
```

In [30]:
```
logistic_model= LogisticRegression()
logistic_model.fit(x_train_scaled, y_train)
predictions=logistic_model.predict(x_test_scaled)
predictions
```

Out[30]:
```
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, 7, 5, 5, 6, 5, 5, 6, 5, 6, 5, 6, 6, 6, 6, 5, 5, 6, 5,
       6, 6, 7, 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, 6, 5, 6, 5,
       7, 5, 6, 5, 6, 6, 6, 5, 5, 6, 6, 5, 6, 5, 5, 5, 6, 6, 5, 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, 6, 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, 6, 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, 6, 5, 5, 5, 6, 5, 5, 6, 5], dtype=int64)
```

In [31]:
```
accuracy= accuracy_score(y_test, predictions)
accuracy
```

Out[31]:
```
0.575
```

In [32]:
```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

In [33]:
```
MSE = mean_squared_error(y_test, predictions)
MAE = mean_absolute_error(y_test, predictions)
MSE, MAE
```

Out[33]: (0.490625, 0.446875)

In [34]:
```
logistic_model=LogisticRegression()
```

In [35]:
```
x=df.drop(['quality'],axis=1)
y=y=df['quality']

x_train, x_test, y_train, y_test= train_test_split(x,y, test_size=0.2, random_s
scaler = preprocessing.StandardScaler().fit(x_train)

x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled=scaler.transform(x_test)

logistic_model= LogisticRegression()
logistic_model.fit(x_train_scaled, y_train)
```

Out[35]: LogisticRegression()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [36]:
```
#paramater grid
param_grid = {
#     'C' [1.5]
     'penalty':['l2','l1','elasticnet'],
    'dual':[False, True],
    'fit_intercept':[True,False],
#     intercept_scaling=1,
    'solver':['liblinear','sag','saga','lbfgs', 'newton-cg'],
    'n_jobs':[-1]

}
```

In [37]:
```
grid_search=GridSearchCV(logistic_model,param_grid,cv=6)
grid_search.fit(x_train,y_train)
```

Out[37]:
```
GridSearchCV(cv=6, estimator=LogisticRegression(),
             param_grid={'dual': [False, True], 'fit_intercept': [True, Fals
e],
                         'n_jobs': [-1], 'penalty': ['l2', 'l1', 'elasticne
t'],
                         'solver': ['liblinear', 'sag', 'saga', 'lbfgs',
                                    'newton-cg']})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [38]:
```
best_params= grid_search.best_params_
print(best_params)
```

```
{'dual': False, 'fit_intercept': True, 'n_jobs': -1, 'penalty': 'l2', 'solve
r': 'newton-cg'}
```

In [39]:
```
best_model=LogisticRegression(**best_params)
best_model.fit(x_train, y_train)
```

Out[39]:
```
LogisticRegression(n_jobs=-1, solver='newton-cg')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [40]:
```
y_pred=best_model.predict(x_test)
accuracy=accuracy_score(y_test, y_pred)
accuracy
```

Out[40]:
```
0.571875
```

In [ ]:

In [ ]: