centra2

March 20, 2024

[63]:
```
#importing the necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler
```

[64]:
```
# loading my data
data=pd.read_csv('C:\\Users\\lynda\\Desktop\\apple_quality.csv')
data.head()
```

[64]:

	A_id	Size	Weight	Sweetness	Crunchiness	Juiciness	Ripeness	\
0	0.0	-3.970049	-2.512336	5.346330	-1.012009	1.844900	0.329840	
1	1.0	-1.195217	-2.839257	3.664059	1.588232	0.853286	0.867530	
2	2.0	-0.292024	-1.351282	-1.738429	-0.342616	2.838636	-0.038033	
3	3.0	-0.657196	-2.271627	1.324874	-0.097875	3.637970	-3.413761	
4	4.0	1.364217	-1.296612	-0.384658	-0.553006	3.030874	-1.303849	

	Acidity	Quality
0	-0.491590483	good
1	-0.722809367	good
2	2.621636473	bad
3	0.790723217	good
4	0.501984036	good

[65]:
```
# converting categorical data to numerica
data['Quality']= data['Quality'].astype('category')
data['Quality']= data['Quality'].cat.codes
data.head()
```

[65]:

	A_id	Size	Weight	Sweetness	Crunchiness	Juiciness	Ripeness	\
0	0.0	-3.970049	-2.512336	5.346330	-1.012009	1.844900	0.329840	
1	1.0	-1.195217	-2.839257	3.664059	1.588232	0.853286	0.867530	
2	2.0	-0.292024	-1.351282	-1.738429	-0.342616	2.838636	-0.038033	
3	3.0	-0.657196	-2.271627	1.324874	-0.097875	3.637970	-3.413761	
4	4.0	1.364217	-1.296612	-0.384658	-0.553006	3.030874	-1.303849	

	Acidity	Quality

```
0  -0.491590483        1
1  -0.722809367        1
2   2.621636473        0
3   0.790723217        1
4   0.501984036        1
```

[66]:
```
# findig where my data has null value
data.isnull().sum()
```

[66]:
```
A_id           1
Size           1
Weight         1
Sweetness      1
Crunchiness    1
Juiciness      1
Ripeness       1
Acidity        0
Quality        0
dtype: int64
```

[67]:
```
#dropping the null data
data = data.dropna()
data.head()
```

[67]:
```
   A_id      Size    Weight  Sweetness  Crunchiness  Juiciness  Ripeness  \
0   0.0 -3.970049 -2.512336   5.346330    -1.012009   1.844900  0.329840
1   1.0 -1.195217 -2.839257   3.664059     1.588232   0.853286  0.867530
2   2.0 -0.292024 -1.351282  -1.738429    -0.342616   2.838636 -0.038033
3   3.0 -0.657196 -2.271627   1.324874    -0.097875   3.637970 -3.413761
4   4.0  1.364217 -1.296612  -0.384658    -0.553006   3.030874 -1.303849

        Acidity  Quality
0  -0.491590483        1
1  -0.722809367        1
2   2.621636473        0
3   0.790723217        1
4   0.501984036        1
```

[68]:
```
# spliting my data into independent and dependent variables
x= data.drop(columns='Quality')
x.head()
```

[68]:
```
   A_id      Size    Weight  Sweetness  Crunchiness  Juiciness  Ripeness  \
0   0.0 -3.970049 -2.512336   5.346330    -1.012009   1.844900  0.329840
1   1.0 -1.195217 -2.839257   3.664059     1.588232   0.853286  0.867530
2   2.0 -0.292024 -1.351282  -1.738429    -0.342616   2.838636 -0.038033
3   3.0 -0.657196 -2.271627   1.324874    -0.097875   3.637970 -3.413761
```

```
4   4.0  1.364217 -1.296612  -0.384658    -0.553006   3.030874 -1.303849

        Acidity
0  -0.491590483
1  -0.722809367
2   2.621636473
3   0.790723217
4   0.501984036
```

[69]:
```
#dependent variable
y=data['Quality']
y.head()
```

[69]:
```
0    1
1    1
2    0
3    1
4    1
Name: Quality, dtype: int8
```

[70]:
```
# Spliting the data into training and testing sets
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.2,␣
 ↪random_state=42)
```

[71]:
```
# Ordinary Logistic Regression
lr = LogisticRegression(max_iter=1000)
lr.fit(x_train, y_train)
```

[71]:
```
LogisticRegression(max_iter=1000)
```

[89]:
```
# finding my y_pred and accuracy for ordinar#
y_pred =lr.predict(x_test)
accuracy_ordinary = accuracy_score(y_test,y_pred)
print('y_pred:',y_pred)
print('accuracy_ordinary',accuracy_ordinary)
```

```
y_pred: [1 1 1 0 0 1 0 0 0 0 1 1 0 1 1 0 0 0 1 0 0 0 1 1 1 0 0 1 1 0 1 0 0 0 1 1
0
 1 0 0 0 0 0 1 0 1 1 1 1 0 1 0 1 1 1 0 0 0 1 1 0 1 0 1 0 0 1 0 1 1 1 0 1 0
 1 1 1 0 0 1 0 1 0 1 0 0 0 0 1 1 0 0 1 1 0 0 1 0 1 1 0 1 0 0 0 0 1 0 0 0 1
 0 0 0 0 0 0 0 0 1 0 0 1 1 0 0 0 1 1 1 0 1 1 1 1 1 0 0 1 0 1 1 1 0 1 0 0 0
 0 1 0 0 0 1 1 0 1 0 0 0 0 1 0 0 1 0 0 1 0 1 0 1 0 1 0 0 1 0 1 0 1 0 0 0 0
 1 1 0 0 0 1 1 0 0 0 1 0 1 0 1 0 0 1 1 0 0 0 0 0 0 1 0 1 1 0 1 1 1 1 1 1 1
 1 0 0 1 1 0 0 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 0 0 0 0 0 0 1 0 0 1 0 1 0 0 0
 0 0 0 0 0 0 0 1 1 0 0 0 1 0 0 0 1 0 1 0 0 1 0 0 0 1 0 0 0 0 0 1 0 1 0 0 1
 0 1 0 0 0 1 1 1 1 1 1 1 0 1 1 1 1 0 0 0 1 1 1 0 1 1 1 0 1 1 1 0 0 0 0 1 0
 0 1 1 0 0 0 0 0 0 1 0 0 1 1 0 1 1 0 1 0 0 1 1 1 1 0 1 1 1 0 1 1 1 0 0 1 1
 1 1 1 1 0 0 1 1 0 0 1 1 1 1 1 1 0 0 1 1 0 1 0 1 0 1 0 1 1 0 1 0 1 0 1 1 1 1
```

```
 1 0 1 0 1 0 1 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 1 0 1 0 0 0 1 1 1 1 0 1 1
 1 0 1 0 1 0 0 1 0 0 0 0 1 1 0 0 0 1 1 0 1 0 1 1 1 0 0 0 1 1 1 0 0 1 0 1 0
 0 0 1 0 1 0 1 1 0 0 0 0 0 0 0 1 0 1 1 0 1 1 1 0 1 1 1 1 1 0 1 0 1 1 0 0 1 1
 1 1 1 1 1 1 1 0 0 1 1 0 1 0 1 1 1 0 1 1 0 0 1 1 0 1 0 0 1 1 0 0 0 1 1 0 1
 1 0 0 0 0 1 0 0 1 1 1 1 1 0 1 0 1 1 0 1 0 1 0 1 0 0 0 1 0 1 1 0 0 0 1 1 1
 1 1 1 0 1 1 1 1 0 1 1 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 1 0 0 1
 0 1 1 0 1 0 0 1 0 1 0 1 1 1 0 0 0 1 1 1 0 0 1 1 1 1 0 1 0 0 1 0 1 1 1 1 0
 1 0 0 1 0 0 1 1 0 0 0 1 0 0 0 1 1 1 0 0 1 0 0 1 0 1 1 1 1 0 1 0 0 0 0 1 0
 0 1 1 0 1 0 0 0 0 1 1 1 0 1 0 1 1 0 1 1 0 0 0 1 1 0 0 0 0 1 0 0 1 1 0 0 1
 1 1 0 1 0 1 1 0 0 1 0 1 0 0 0 0 0 0 1 1 0 1 1 1 1 1 1 0 0 0 1 1 0 1 1 0 0
 1 0 0 1 1 1 1 1 0 0 1 0 0 1 0 0 0 0 1 1 1 1 1]
accuracy_ordinary 0.75625
```

[76]:
```
#scaler
scaler = StandardScaler()
x_train_opt=scaler.fit_transform(x_train)
x_test_opt =scaler.transform(x_test)
```

[77]:
```
# Building model for optimized LogisticRegression
lr_opt = LogisticRegression(random_state=0,
                            max_iter =1000,
                           C=1,
                            fit_intercept =True

                           ).fit(x_train,y_train).fit(x_train,y_train)
```

[90]:
```
# finding my y_pred and accuracy for ordinar#
y_pred_opt =lr_opt.predict(x_test_opt)
accuracy_opt = accuracy_score(y_test,y_pred)
print('y_pred_opt:',y_pred_opt)
print('accuracy_opt',accuracy_opt)
```

```
y_pred_opt: [1 1 1 1 0 1 0 1 1 1 1 1 0 1 1 0 0 0 1 0 0 0 1 1 1 1 0 1 1 0 1 1 0 1
1 1 0
 1 1 1 0 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 0 0 1 0 1 0 1 1 1 0 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 0 0 1 0 1 1 1
 1 0 0 1 0 1 0 1 1 0 0 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1
 1 1 0 0 1 1 1 0 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 0 0 1 1
 1 1 0 0 0 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 1
 1 0 0 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 0 0 1 1 1 1 0 1 0 1 1
 0 1 0 0 0 0 0 1 1 1 1 1 1 0 0 0 1 1 1 1 0 1 0 0 0 1 1 1 0 1 1 1 0 1 0 1 1
 0 1 0 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 1 0 1 1
 1 1 1 0 0 0 0 0 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1
 1 1 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1
 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 0 1 1 1 1 0 0 1 1 1 1 1 1
 1 1 1 1 1 1 0 1 1 0 1 0 1 1 0 0 1 1 1 0 1 0 1 1 1 1 0 0 1 1 1 1 0 1 0 1 0
 1 0 1 1 1 0 1 1 1 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1
 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0 0 1 1 0 1 0 0 1 1 1 0 1 1 1 0 1
```

```
 1 1 0 0 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 0 0 1 0 1 1 0 0 1 1 1 1
 1 1 1 0 1 1 1 1 0 1 1 0 0 0 1 0 1 1 0 0 0 1 1 1 0 0 0 1 0 0 1 0 1 1 0 0 1
 0 1 1 0 1 1 1 1 0 1 1 1 1 1 0 1 0 1 1 1 1 0 1 1 1 1 0 1 0 0 1 0 1 1 1 1 0
 1 0 0 1 1 1 1 1 0 1 0 1 0 1 0 1 1 1 0 1 1 0 1 1 1 1 1 1 1 0 1 0 0 0 0 1 0
 0 1 1 1 1 0 0 1 1 1 1 1 0 1 0 1 1 0 1 1 1 0 0 1 1 1 0 1 1 1 0 1 1 1 1 0 1
 1 1 1 1 0 1 1 0 1 1 0 1 1 1 0 0 0 1 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 0 0
 1 0 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1]
accuracy_opt 0.75625

C:\Users\lynda\anaconda3\Lib\site-packages\sklearn\base.py:464: UserWarning: X
does not have valid feature names, but LogisticRegression was fitted with
feature names
  warnings.warn(
```

[86]:
```
if accuracy_ordinary>accuracy_opt:
    print('The ordinary model performs better than the optimized model')
elif accuracy_ordinary < accuracy_opt:
    print('The ordinary model performs better than the optimized')
else:
    print('Both modes have the same accuracy')
```

```
Both modes have the same accuracy
```

[ ]:

[ ]: