1. Use your own dataset (`Heart.csv` is acceptable), create a train and a test set, and build 2 models: Logistic Regression and Decision Tree (shallow). Compare the test results using `classification_report` and `confusion_matrix`. Explain which algorithm is optimal

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.linear_model import LogisticRegression
from sklearn import preprocessing
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
```

I used a dataset from UCI: Wine Quality (Red wine)

(https://archive.ics.uci.edu/dataset/186/wine+quality (https://archive.ics.uci.edu/dataset/186/wine+quality))

- This data shows scores of wine quality(from 1 to 10) with 11 related features
- Good wine : score >= 6, normal wine : score < 6

1-A. Preprocessing

In [2]:
```
# read the dataset as 'df'
df = pd.read_csv("winequality-red.csv")

df.head()
```

Out[2]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

- As variable 'quality' has values from 1 to 10, transforming the values into binary values is necessary
- values from 6 to 10 are transformed into value 1(good wine), and values from 1 to 5 are transformed into value 0(normal wine)

In [3]:
```
# transform values from 6 to 10 into value 1
df['quality'] = df.quality.between(6,10).astype(int)
df.head()
```

Out[3]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	0
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	0
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	0
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	1
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	0

In [4]:
```
#do train/test split

x_train, x_test, y_train, y_test = train_test_split(df.drop(['quality'], axis=1), df.quality, test_size=.

x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

Out[4]:
```
((1279, 11), (320, 11), (1279,), (320,))
```

In [5]:
```
y_test.value_counts()
```

Out[5]:
```
quality
1    179
0    141
Name: count, dtype: int64
```

1-B. Logistic Regression

In [6]:
```
#generate logistic regression model as 'model1'

model1 = LogisticRegression()
model1.fit(x_train, y_train)
```

```
C:WUsersWkcosmWanaconda3WLibWsite-packagesWsklearnWlinear_modelW_logistic.py:469: ConvergenceWarning: lb
fgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/
preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.
org/stable/modules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

Out[6]:
```
LogisticRegression
LogisticRegression()
```

In [7]:
```
#As the warning sign indicates total number of iterations reached limit,
#I increased maximum number of iterations from 100(defalut) to 1000

model1 = LogisticRegression(max_iter=1000)
model1.fit(x_train, y_train)
```

Out[7]:
```
LogisticRegression
LogisticRegression(max_iter=1000)
```

In [8]:
```
#Print the test results
prediction1 = model1.predict(x_test)
print(classification_report(y_test, prediction1))
print(confusion_matrix(y_test, prediction1))
```

```
              precision    recall  f1-score   support

           0       0.70      0.74      0.72       141
           1       0.79      0.74      0.76       179

    accuracy                           0.74       320
   macro avg       0.74      0.74      0.74       320
weighted avg       0.75      0.74      0.74       320

[[105  36]
 [ 46 133]]
```

1-C. Decision Tree

In [9]:
```
#build a shallow decision tree model(max_depth=3) as 'model2'

model2 = DecisionTreeClassifier(criterion='entropy', max_depth=3)
model2.fit(x_train, y_train)
```

Out[9]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=3)
```

In [10]:
```
#Print the test results
prediction2 = model2.predict(x_test)
print(classification_report(y_test, prediction2))
print(confusion_matrix(y_test, prediction2))
```

```
              precision    recall  f1-score   support

           0       0.58      0.92      0.71       141
           1       0.89      0.47      0.62       179

    accuracy                           0.67       320
   macro avg       0.73      0.70      0.67       320
weighted avg       0.75      0.67      0.66       320

[[130  11]
 [ 94  85]]
```

Result : Comparing cell 8 and 10, Logistic Regression model is more optimal.

- Logistic Regression model shows higher f1-score in predicting good wine (value 1, 0.76 > 0.62) and normal wine(value 0, 0.72 > 0.71)

2. Repeat 1. but let the Decision Tree be much deeper to allow over-fitting. Compare the two models' test results again, and explain which is optimal

2-A. Deeper Decision Tree(max_depth=10)

In [11]:
```
#build a deeper decision tree model(max_depth=10) as 'model3'

model3 = DecisionTreeClassifier(criterion='entropy', max_depth=10)
model3.fit(x_train, y_train)
```

Out[11]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=10)
```

In [12]:
```
#Print the test results
prediction3 = model3.predict(x_test)
print(classification_report(y_test, prediction3))
print(confusion_matrix(y_test, prediction3))
```

```
              precision    recall  f1-score   support

           0       0.71      0.75      0.73       141
           1       0.79      0.75      0.77       179

    accuracy                           0.75       320
   macro avg       0.75      0.75      0.75       320
weighted avg       0.76      0.75      0.75       320

[[106  35]
 [ 44 135]]
```

2-B. Much Deeper Decision Tree(max_depth=50)

In [13]:
```
#build a much deeper decision tree model(max_depth=50) as 'model4'

model4 = DecisionTreeClassifier(criterion='entropy', max_depth=50)
model4.fit(x_train, y_train)
```

Out[13]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=50)
```

In [14]:
```
#Print the test results
prediction4 = model4.predict(x_test)
print(classification_report(y_test, prediction4))
print(confusion_matrix(y_test, prediction4))
```

```
              precision    recall  f1-score   support

           0       0.70      0.79      0.74       141
           1       0.82      0.73      0.77       179

    accuracy                           0.76       320
   macro avg       0.76      0.76      0.76       320
weighted avg       0.77      0.76      0.76       320

[[112  29]
 [ 48 131]]
```

Result : Comparing the cell 8, 12, and 14, deeper Decision Tree model outperforms Logistic Regression model

- both Decision Tree model shows higher f1-score than Logistic Regression model
- f1-score improved as max depth of Decision Tree increased