ML_with_SKLearn

November 6, 2022

1 Machine Learning in Python with SKLearn

By: Jonathan Blade

1.1 Reading the data in:

[14]:
```
import pandas as pd

carData = pd.read_csv('/content/drive/MyDrive/Data/Auto.csv')
print(carData.head())
```

```
    mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
0  18.0          8         307.0         130    3504          12.0  70.0
1  15.0          8         350.0         165    3693          11.5  70.0
2  18.0          8         318.0         150    3436          11.0  70.0
3  16.0          8         304.0         150    3433          12.0  70.0
4  17.0          8         302.0         140    3449           NaN  70.0

   origin                       name
0       1  chevrolet chevelle malibu
1       1          buick skylark 320
2       1         plymouth satellite
3       1              amc rebel sst
4       1                ford torino
```

1.2 Data Exploration:

[ ]:
```
carData.mpg.describe()
```

[ ]:
```
count    392.000000
mean      23.445918
std        7.805007
min        9.000000
25%       17.000000
50%       22.750000
```

```
75%        29.000000
max        46.600000
Name: mpg, dtype: float64
```

[ ]:
```
carData.year.describe()
```

[ ]:
```
count    390.000000
mean      76.010256
std        3.668093
min       70.000000
25%       73.000000
50%       76.000000
75%       79.000000
max       82.000000
Name: year, dtype: float64
```

[ ]:
```
carData.weight.describe()
```

[ ]:
```
count     392.000000
mean     2977.584184
std       849.402560
min      1613.000000
25%      2225.250000
50%      2803.500000
75%      3614.750000
max      5140.000000
Name: weight, dtype: float64
```

The describe function tells us the ranges and averages of each column.
MPG: - Range: 9.0 - 46.0 - Average: 23.4

Year:
- Range: 70.0 - 82.0 - Average: 76.0

Weight:
- Range: 1613.0 - 5140.0 - Average: 2977.6

1.3 Analyzing Data Types:

[ ]:
```
carData.dtypes
```

[ ]:
```
mpg             float64
cylinders         int64
displacement    float64
horsepower        int64
weight            int64
acceleration    float64
```

```
year              float64
origin              int64
name               object
dtype: object
```

[6]:
```
carData.cylinders = carData.cylinders.astype('category').cat.codes
```

[5]:
```
carData.origin = carData.origin.astype('category')
```

[7]:
```
carData.dtypes
```

[7]:
```
mpg              float64
cylinders           int8
displacement     float64
horsepower         int64
weight             int64
acceleration     float64
year             float64
origin          category
name              object
dtype: object
```

1.4 Dealing With N/A's:

[15]:
```
carData.isnull().sum()
```

[15]:
```
mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    1
year            2
origin          0
name            0
dtype: int64
```

[16]:
```
print('\nDimensions of data:', carData.shape)
carData = carData.dropna()
print('\nDimensions of data after removing N/A\'s:', carData.shape)
```

```

Dimensions of data: (392, 9)

Dimensions of data after removing N/A's: (389, 9)
```

1.5 Modifying Columns:

[44]:
```
mpgAvg = carData.mpg.mean()
mpgHigh = []

counter = 0

for x in carData.mpg:
  if x > mpgAvg:
    mpgHigh.append(1)
  else:
    mpgHigh.append(0)

carData['mpg_high'] = mpgHigh

carData = carData.drop('mpg', axis=1)

print(carData.head())
```

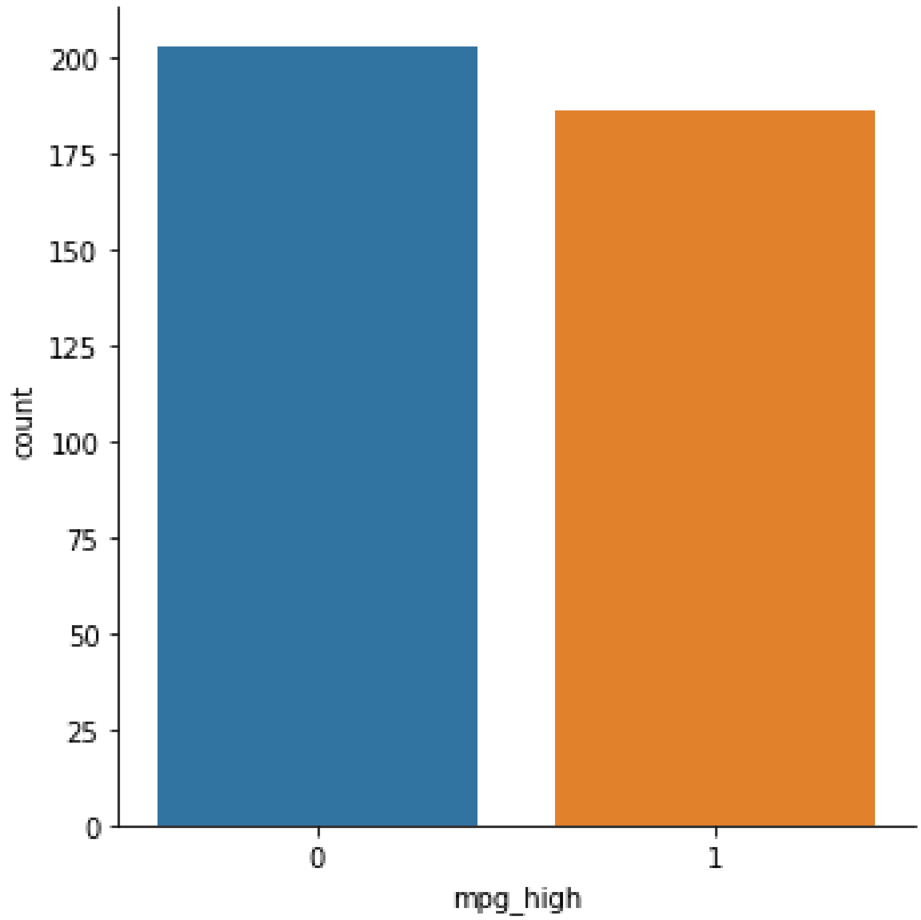
```
   cylinders  displacement  horsepower  weight  acceleration  year  origin  \
0          8         307.0         130    3504          12.0  70.0       1
1          8         350.0         165    3693          11.5  70.0       1
2          8         318.0         150    3436          11.0  70.0       1
3          8         304.0         150    3433          12.0  70.0       1
6          8         454.0         220    4354           9.0  70.0       1

                        name  mpg_high
0  chevrolet chevelle malibu         0
1          buick skylark 320         0
2         plymouth satellite         0
3              amc rebel sst         0
6           chevrolet impala         0
```

1.6 Exploring the Data with Graphs:

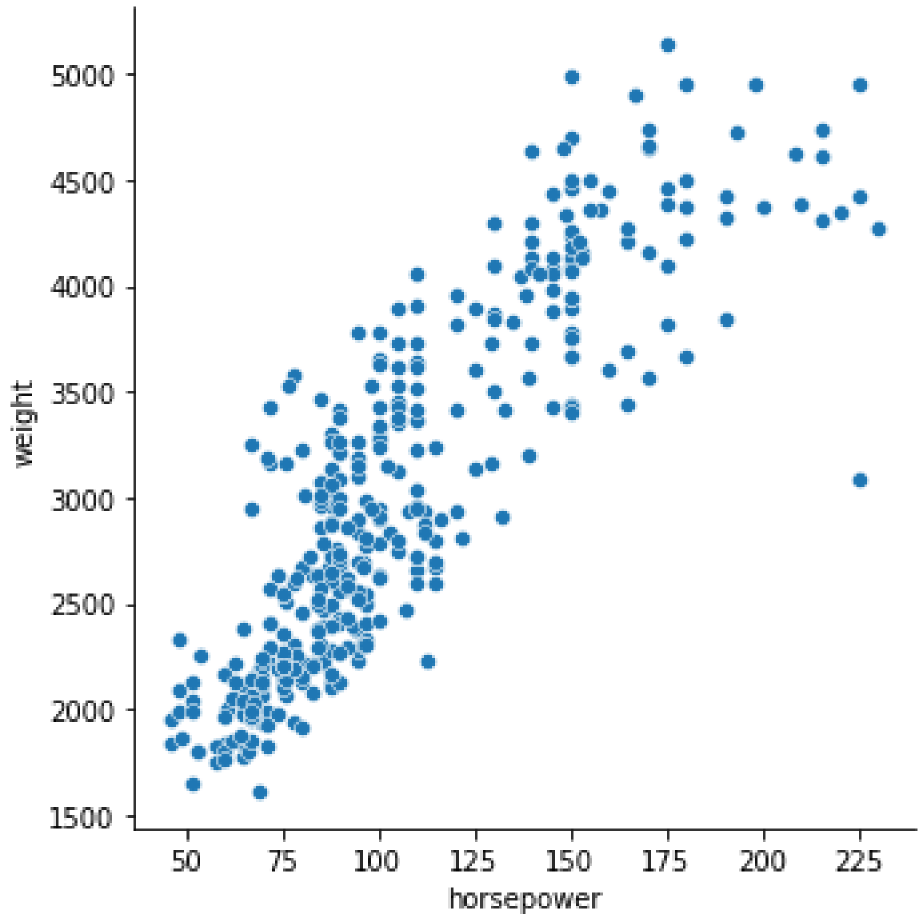
[49]:
```
import seaborn as sb

graph = sb.catplot(x="mpg_high", kind='count', data=carData)
```



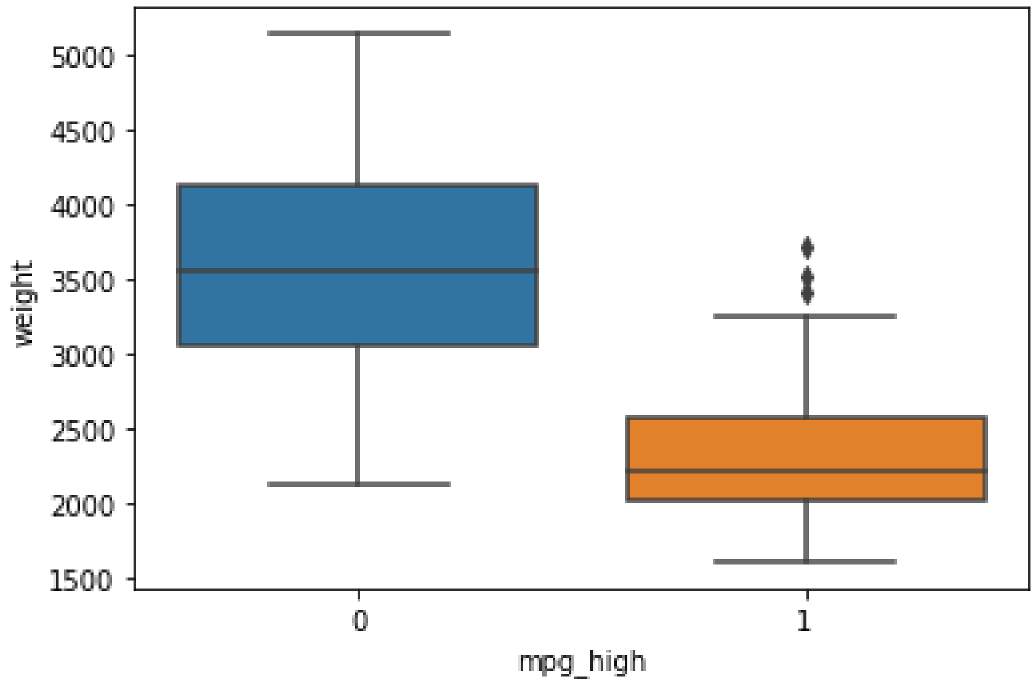
We can see from this graph that there are slightly more cars with below average MPG than there are cars with above average MPG.

[50]:
```
graph2 = sb.relplot(x="horsepower", y="weight", data=carData)
```



We can deterime from this graph that there appears to be a positive correlation between weight and horsepower.

[52]:
```
graph3 = sb.boxplot(x='mpg_high', y='weight', data=carData)
```



This boxplot shows that cars with high MPG tend to weigh less than cars with low MPG.

1.7 Splitting the Data:

[54]:
```
from sklearn.model_selection import train_test_split

X = carData.iloc[:, 0:7]
y = carData.iloc[:, 8]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=1234)

print('Dimensions of training data, X: ', X_train.shape, ' y: ', y_train.shape)
print('Dimensions of test data, X: ', X_test.shape, ' y: ', y_test.shape)
```

```
Dimensions of training data, X:  (311, 7)  y:  (311,)
Dimensions of test data, X:  (78, 7)  y:  (78,)
```

1.8 Logistic Regession:

[62]:
```
from sklearn.linear_model import LogisticRegression

clf1 = LogisticRegression(solver='lbfgs', max_iter=1000)
clf1.fit(X_train, y_train)
clf1.score(X_train, y_train)
```

[62]: 0.9067524115755627

[63]:
```
pred1 = clf1.predict(X_test)
```

[64]:
```
from sklearn.metrics import classification_report

print(classification_report(y_test, pred1))
```

```
              precision    recall  f1-score   support

           0       0.98      0.82      0.89        50
           1       0.75      0.96      0.84        28

    accuracy                           0.87        78
   macro avg       0.86      0.89      0.87        78
weighted avg       0.89      0.87      0.87        78
```

1.9 Decision Tree:

[65]:
```
from sklearn.tree import DecisionTreeClassifier

clf2 = DecisionTreeClassifier()
clf2.fit(X_train, y_train)
```

[65]: DecisionTreeClassifier()

[66]:
```
pred2 = clf2.predict(X_test)
```

[67]:
```
print(classification_report(y_test, pred2))
```

```
              precision    recall  f1-score   support

           0       0.92      0.90      0.91        50
           1       0.83      0.86      0.84        28

    accuracy                           0.88        78
   macro avg       0.87      0.88      0.88        78
```

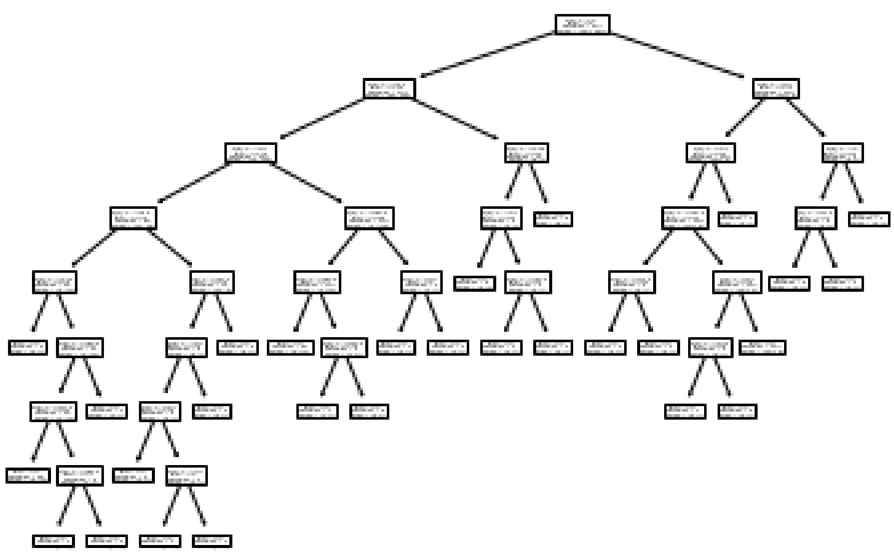
```
weighted avg       0.89      0.88      0.89        78
```

[70]:
```python
from sklearn import tree
tree.plot_tree(clf2)
```

[70]:
```
[Text(0.6507352941176471, 0.9444444444444444, 'X[0] <= 5.5\ngini = 0.5\nsamples
= 311\nvalue = [153, 158]'),
 Text(0.4338235294117647, 0.8333333333333334, 'X[2] <= 101.0\ngini =
0.239\nsamples = 173\nvalue = [24, 149]'),
 Text(0.27941176470588236, 0.7222222222222222, 'X[5] <= 75.5\ngini =
0.179\nsamples = 161\nvalue = [16, 145]'),
 Text(0.14705882352941177, 0.6111111111111112, 'X[1] <= 119.5\ngini =
0.362\nsamples = 59\nvalue = [14, 45]'),
 Text(0.058823529411764705, 0.5, 'X[4] <= 13.75\ngini = 0.159\nsamples =
46\nvalue = [4, 42]'),
 Text(0.029411764705882353, 0.3888888888888889, 'gini = 0.0\nsamples = 2\nvalue
= [2, 0]'),
 Text(0.08823529411764706, 0.3888888888888889, 'X[3] <= 2683.0\ngini =
0.087\nsamples = 44\nvalue = [2, 42]'),
 Text(0.058823529411764705, 0.2777777777777778, 'X[3] <= 2377.0\ngini =
0.045\nsamples = 43\nvalue = [1, 42]'),
 Text(0.029411764705882353, 0.16666666666666666, 'gini = 0.0\nsamples =
38\nvalue = [0, 38]'),
 Text(0.08823529411764706, 0.16666666666666666, 'X[3] <= 2385.0\ngini =
0.32\nsamples = 5\nvalue = [1, 4]'),
 Text(0.058823529411764705, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nvalue
= [1, 0]'),
 Text(0.11764705882352941, 0.05555555555555555, 'gini = 0.0\nsamples = 4\nvalue
= [0, 4]'),
 Text(0.11764705882352941, 0.2777777777777778, 'gini = 0.0\nsamples = 1\nvalue =
[1, 0]'),
 Text(0.23529411764705882, 0.5, 'X[3] <= 2567.0\ngini = 0.355\nsamples =
13\nvalue = [10, 3]'),
 Text(0.20588235294117646, 0.3888888888888889, 'X[5] <= 73.5\ngini =
0.469\nsamples = 8\nvalue = [5, 3]'),
 Text(0.17647058823529413, 0.2777777777777778, 'X[2] <= 88.0\ngini =
0.278\nsamples = 6\nvalue = [5, 1]'),
 Text(0.14705882352941177, 0.16666666666666666, 'gini = 0.0\nsamples = 4\nvalue
= [4, 0]'),
 Text(0.20588235294117646, 0.16666666666666666, 'X[4] <= 17.5\ngini =
0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.17647058823529413, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nvalue
= [0, 1]'),
 Text(0.23529411764705882, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nvalue
= [1, 0]'),
 Text(0.23529411764705882, 0.2777777777777778, 'gini = 0.0\nsamples = 2\nvalue =
```

```
[0, 2]'),
 Text(0.2647058823529412, 0.3888888888888889, 'gini = 0.0\nsamples = 5\nvalue =
[5, 0]'),
 Text(0.4117647058823529, 0.6111111111111112, 'X[3] <= 3250.0\ngini =
0.038\nsamples = 102\nvalue = [2, 100]'),
 Text(0.35294117647058826, 0.5, 'X[3] <= 2880.0\ngini = 0.02\nsamples =
100\nvalue = [1, 99]'),
 Text(0.3235294117647059, 0.3888888888888889, 'gini = 0.0\nsamples = 94\nvalue =
[0, 94]'),
 Text(0.38235294117647056, 0.3888888888888889, 'X[3] <= 2920.0\ngini =
0.278\nsamples = 6\nvalue = [1, 5]'),
 Text(0.35294117647058826, 0.2777777777777778, 'gini = 0.0\nsamples = 1\nvalue =
[1, 0]'),
 Text(0.4117647058823529, 0.2777777777777778, 'gini = 0.0\nsamples = 5\nvalue =
[0, 5]'),
 Text(0.47058823529411764, 0.5, 'X[2] <= 82.5\ngini = 0.5\nsamples = 2\nvalue =
[1, 1]'),
 Text(0.4411764705882353, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
 Text(0.5, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.5882352941176471, 0.7222222222222222, 'X[4] <= 14.45\ngini =
0.444\nsamples = 12\nvalue = [8, 4]'),
 Text(0.5588235294117647, 0.6111111111111112, 'X[5] <= 76.0\ngini =
0.444\nsamples = 6\nvalue = [2, 4]'),
 Text(0.5294117647058824, 0.5, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.5882352941176471, 0.5, 'X[3] <= 2760.0\ngini = 0.444\nsamples = 3\nvalue
= [2, 1]'),
 Text(0.5588235294117647, 0.3888888888888889, 'gini = 0.0\nsamples = 2\nvalue =
[2, 0]'),
 Text(0.6176470588235294, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
 Text(0.6176470588235294, 0.6111111111111112, 'gini = 0.0\nsamples = 6\nvalue =
[6, 0]'),
 Text(0.8676470588235294, 0.8333333333333334, 'X[5] <= 79.5\ngini =
0.122\nsamples = 138\nvalue = [129, 9]'),
 Text(0.7941176470588235, 0.7222222222222222, 'X[4] <= 21.6\ngini =
0.045\nsamples = 129\nvalue = [126, 3]'),
 Text(0.7647058823529411, 0.6111111111111112, 'X[3] <= 2737.0\ngini =
0.031\nsamples = 128\nvalue = [126, 2]'),
 Text(0.7058823529411765, 0.5, 'X[3] <= 2674.0\ngini = 0.444\nsamples = 3\nvalue
= [2, 1]'),
 Text(0.6764705882352942, 0.3888888888888889, 'gini = 0.0\nsamples = 2\nvalue =
[2, 0]'),
 Text(0.7352941176470589, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
 Text(0.8235294117647058, 0.5, 'X[2] <= 83.0\ngini = 0.016\nsamples = 125\nvalue
= [124, 1]'),
```

```
 Text(0.7941176470588235, 0.3888888888888889, 'X[4] <= 18.55\ngini =
0.375\nsamples = 4\nvalue = [3, 1]'),
 Text(0.7647058823529411, 0.2777777777777778, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
 Text(0.8235294117647058, 0.2777777777777778, 'gini = 0.0\nsamples = 3\nvalue =
[3, 0]'),
 Text(0.8529411764705882, 0.3888888888888889, 'gini = 0.0\nsamples = 121\nvalue
= [121, 0]'),
 Text(0.8235294117647058, 0.6111111111111112, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
 Text(0.9411764705882353, 0.7222222222222222, 'X[6] <= 1.5\ngini =
0.444\nsamples = 9\nvalue = [3, 6]'),
 Text(0.9117647058823529, 0.6111111111111112, 'X[1] <= 247.0\ngini =
0.48\nsamples = 5\nvalue = [3, 2]'),
 Text(0.8823529411764706, 0.5, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.9411764705882353, 0.5, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.9705882352941176, 0.6111111111111112, 'gini = 0.0\nsamples = 4\nvalue =
[0, 4]')]
```



1.10 Neural Network:

[71]:
```
from sklearn import preprocessing

scaler = preprocessing.StandardScaler().fit(X_train)

X_scaledTrain = scaler.transform(X_train)
```

```python
X_scaledTest = scaler.transform(X_test)
```

[72]:
```python
from sklearn.neural_network import MLPClassifier

clf3 = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5,2), max_iter=500,␣
↪random_state=1234)
clf3.fit(X_scaledTrain, y_train)
```

[72]:
```
MLPClassifier(hidden_layer_sizes=(5, 2), max_iter=500, random_state=1234,
              solver='lbfgs')
```

[78]:
```python
pred3 = clf3.predict(X_scaledTest)
print(classification_report(y_test, pred3))
```

```
              precision    recall  f1-score   support

           0       0.94      0.88      0.91        50
           1       0.81      0.89      0.85        28

    accuracy                           0.88        78
   macro avg       0.87      0.89      0.88        78
weighted avg       0.89      0.88      0.89        78
```

[86]:
```python
clf4 = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(6,3,2), max_iter=500,␣
↪random_state=1234)
clf4.fit(X_scaledTrain, y_train)
```

[86]:
```
MLPClassifier(hidden_layer_sizes=(6, 3, 2), max_iter=500, random_state=1234,
              solver='lbfgs')
```

[87]:
```python
pred4 = clf4.predict(X_scaledTest)
print(classification_report(y_test, pred4))
```

```
              precision    recall  f1-score   support

           0       0.98      0.88      0.93        50
           1       0.82      0.96      0.89        28

    accuracy                           0.91        78
   macro avg       0.90      0.92      0.91        78
weighted avg       0.92      0.91      0.91        78
```

Both networks performed about the same, with the second network having slightly better results. I believe what the lead the second network to perform better was the additional hidden layer as well as a larger first hidden layer. However, one concern is overfitting the data which can occur with neural networks and small datasets.

1.11 Analysis:

The results of each algorithm are roughly comparable, with the second neural network competing just slightly better than the others.
Overall, precision for class 0 was over 0.9 for each algorithim with the highest being Logistic Regression and the second neural network, both with 0.98. However, the decision tree algorithm had the best values for both recall in class 0 and precision in class 1. The second neural network had the best recall for class 1, again tying with the logistic regression model. The second neural network manages to have the highest accuracy despite tying with the logistic regression model for both of its highest values.
Overall, the second neural network performed the best. This could be due to the neural network overfitting the datam which becomes increasingly likely the smaller the dataset.
Overall, I think SKLearn feels easier to use and simpler than the equivalent functions in R. I still have a slight preference for R but possibly just becasue I am more familiar with it than SKLearn.