Machine Learning with sklearn

Abigail Solomon

1. Read the Auto data

In [ ]:
```
# Use pandas to read the data
import pandas as pd
from google.colab import drive
drive.mount('/content/ML_drive')
```

In [410]:
```
df = pd.read_csv('/content/ML_drive/MyDrive/Auto.csv')
```

In [411]:
```
# Out put the first few rows
print(df.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  
```

In [412]:
```
# Out put the dimensions of the data
print('\nDimensions of the data: ', df.shape)
```

```
Dimensions of the data:  (392, 9)
```

2. Data Exploration

In [413]:
```
# Use describe() on the mpg,weight, and year columns
df[['mpg','weight','year']].describe()
```

Out[413]:

	mpg	weight	year
count	392.000000	392.000000	390.000000
mean	23.445918	2977.584184	76.010256
std	7.805007	849.402560	3.668093
min	9.000000	1613.000000	70.000000
25%	17.000000	2225.250000	73.000000
50%	22.750000	2803.500000	76.000000
75%	29.000000	3614.750000	79.000000
max	46.600000	5140.000000	82.000000

In [414]:
```
# Lets find the range for each column using numpy
import numpy as np
range = np.max(df.loc[:, ['mpg', 'weight', 'year']]) - np.min(df.loc[:, ['mpg'
, 'weight', 'year']])
print('Range of mpg, weight, and year:\n', range)
```

```
Range of mpg, weight, and year:
 mpg          37.6
weight    3527.0
year        12.0
dtype: float64
```

Comments indicating the range and average of each column

mpg:

The average is 23.45, which is less than the median value-22.75. It shows that the data is right skewed. The range is between 9 and 46, that is 37. It's found that, the border of the 25% and 75% of the data is 17.0 and 29.0 respectively. It's the second out of the three columns with respect to the range value.

weight:

The average is 2977.58, which is also less than the median value-2308.50. It shows that the data is right skewed. The range is between 1613 and 5140. It's widely spread. It's found that, the border of the 25% and 75% of the data is 2225.25 and 3614.75 respectively. It's the first out of the three columns with respect to the range value, it could be because it has the most number of observations.

year:

The average is 76.01, which is just slightly less than the median value-76.00.The range is between 70 and 82, which is 12. It's found that, the border of the 25% and 75% of the data is 73.0 and 79.0 respectively. It's the least out of the three columns with respect to the range value.

3. Explore data types

In [415]:
```
# Check the data types of all columns
df.dtypes
```

Out[415]:
```
mpg             float64
cylinders         int64
displacement    float64
horsepower        int64
weight            int64
acceleration    float64
year            float64
origin            int64
name             object
dtype: object
```

In [416]:
```python
# Change the cylinders column to categorical (use cat.codes)
df.cylinders = df.cylinders.astype('category').cat.codes
# Change the origin column to categorical (don’t use cat.codes)
df.origin = df.origin.astype('category')
# Verify the changes with the dtypes attribute
df.dtypes
```

Out[416]:
```
mpg              float64
cylinders           int8
displacement     float64
horsepower         int64
weight             int64
acceleration     float64
year             float64
origin          category
name              object
dtype: object
```

4. Deal with NAs

In [417]:
```python
# Delete rows with NAs
df = df.dropna()
# Output the new dimensions
print('\nDimensions of data frame:', df.shape)
```

```
Dimensions of data frame: (389, 9)
```

5. Modify columns

In [418]:

```
# Make a new column, mpg_high, which is categorical: column == 1 if mpg > aver
age mpg, else == 0
import numpy as np
mpg_mean = np.mean(df.mpg)
mpg_high = []
# the column == 1 if mpg > average mpg, else == 0
for item in df.mpg:
    if item > mpg_mean:
        mpg_high += [1]
    else:
        mpg_high += [0]

df = df.assign(mpg_high = mpg_high)
df.mpg_high = df.mpg_high.astype('category')
df = df.drop(columns=['mpg','name'])
df.head()
```

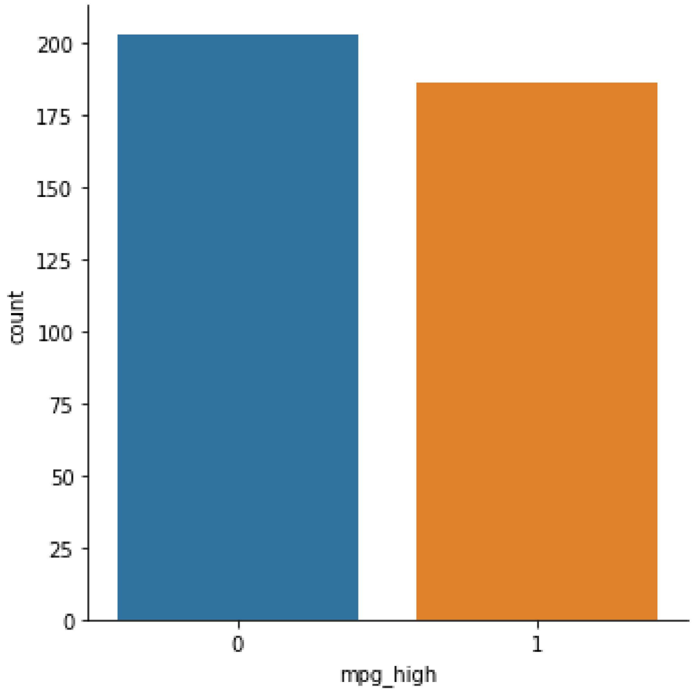
Out[418]:

	cylinders	displacement	horsepower	weight	acceleration	year	origin	mpg_high
0	4	307.0	130	3504	12.0	70.0	1	0
1	4	350.0	165	3693	11.5	70.0	1	0
2	4	318.0	150	3436	11.0	70.0	1	0
3	4	304.0	150	3433	12.0	70.0	1	0
6	4	454.0	220	4354	9.0	70.0	1	0

6. Data exploration with graphs

In [419]:
```python
# Seaborn catplot on the mpg_high column
import seaborn as sb
sb.catplot(x="mpg_high", kind="count", data= df)
```

Out[419]: <seaborn.axisgrid.FacetGrid at 0x7fd874a8a250>

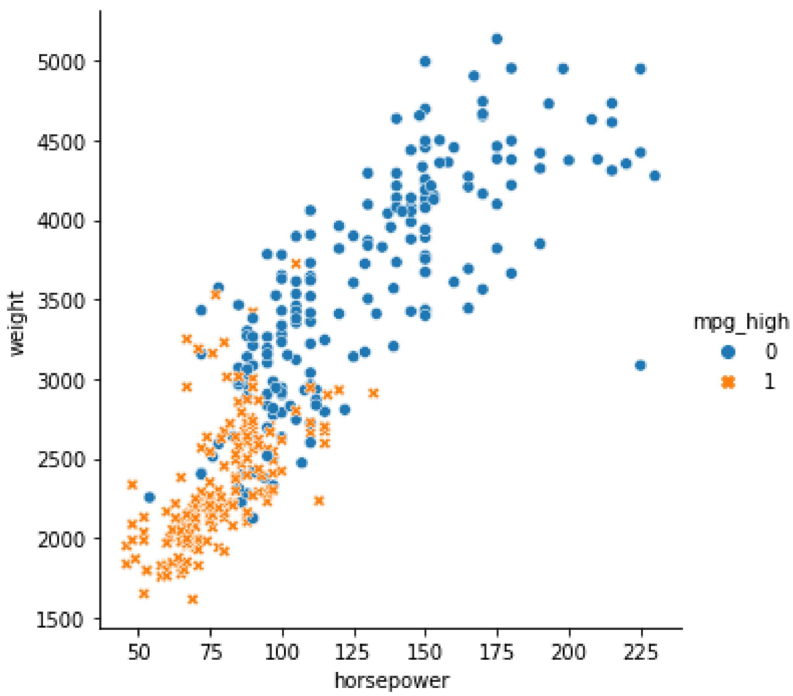


What I learn from the above graph:

The catplot shows the distribution of the target, mpg_high, plots a categorical value, mpg_high, on the x axis, and numerical values, on the y axis. It looks that, there are more cars with a fuel efficiency lower than 23 mpg.

In [420]:
```
# Seaborn relplot with horsepower on the x axis, weight on the y axis, setting hue or style to mpg_high
sb.relplot(x='horsepower', y='weight', data=df, hue=df.mpg_high, style=df.mpg_high)
```

Out[420]: <seaborn.axisgrid.FacetGrid at 0x7fd87497d9d0>

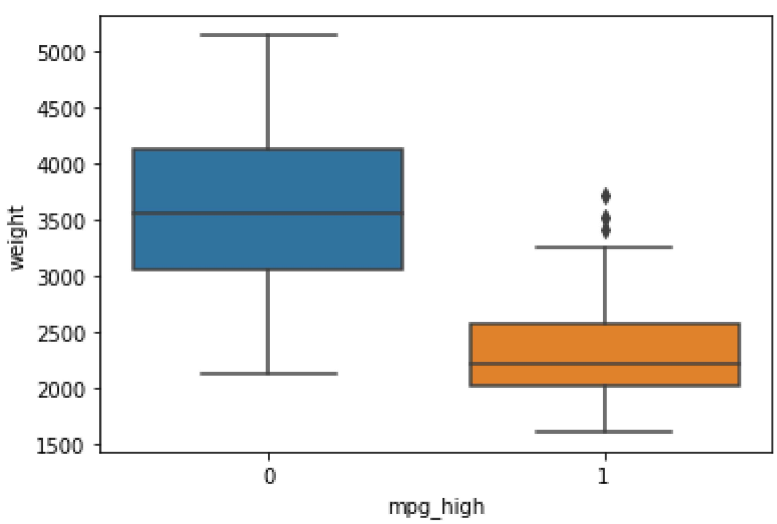


What I learn from the above graph:

The relplot plots relationship. The hue semantic(mpg_high) was categorical, so the default qualitative palette was applied. It shows that there is a high correlation between a car's horsepower and a car's weight

In [421]:
```
# Seaborn boxplot with mpg_high on the x axis and weight on the y axis
sb.boxplot(x='mpg_high', y='weight', data=df)
```

Out[421]: <matplotlib.axes._subplots.AxesSubplot at 0x7fd874899a50>



What I learn from the above graph:

The boxplot represents the depicting of the two groups of numerical data through their quartiles by detecting the outlier in data set. It shows that the weight of the car has a significant impact on the fuel it takes.

7. Train/Test split

In [422]:

```
# Train/Test split 80/20
from sklearn.model_selection import train_test_split

X = df.iloc[:, 0:6]
y = df.iloc[:, 7]


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
m_state=1234)
print('train size:', X_train.shape)
print('test size:', X_test.shape)

# Output the dimensions of train and test
print('\nDimensions of data frame:', df.shape)
```

```
train size: (311, 6)
test size: (78, 6)

Dimensions of data frame: (389, 8)
```

8. Logistic Regression

In [423]:

```
# Train a logistic regression model using solver lbfgs
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
score, log_loss
from sklearn.metrics import classification_report, confusion_matrix

# Use seed 1234 so we all get the same results
clf = LogisticRegression(random_state=1234, solver='lbfgs', max_iter=500)
clf.fit(X_train,y_train)
clf.score(X_train,y_train)

# Test and evaluate
pred = clf.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred))
print('recall score: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))

clfpred = clf.predict(X_test)
# Print metrics using the classification report
print("\n Metrics using the classification report\n")
print(classification_report(y_test, clfpred))
print(" Confusion_matrix results\n")
print(confusion_matrix(y_test, clfpred))
```

```
accuracy score:  0.8589743589743589
precision score:  0.7297297297297297
recall score:  0.9642857142857143
f1 score:  0.8307692307692307

 Metrics using the classification report

              precision    recall  f1-score   support

           0       0.98      0.80      0.88        50
           1       0.73      0.96      0.83        28

    accuracy                           0.86        78
   macro avg       0.85      0.88      0.85        78
weighted avg       0.89      0.86      0.86        78

 Confusion_matrix results

[[40 10]
 [ 1 27]]
```

9. Decision Tree

In [424]:
```
# Train a decision tree
from sklearn.tree import DecisionTreeClassifier,plot_tree

DT = DecisionTreeClassifier(random_state=1234)
DT.fit(X_train,y_train)
DT.score(X_train,y_train)

# Test and evaluate
pred = clf.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred))
print('recall score: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))

DTpred = DT.predict(X_test)
# Print the classification report metrics
print("\n Metrics using the classification report\n")
print(classification_report(y_test, DTpred))
print(" Confusion_matrix results\n")
print(confusion_matrix(y_test, DTpred))
```

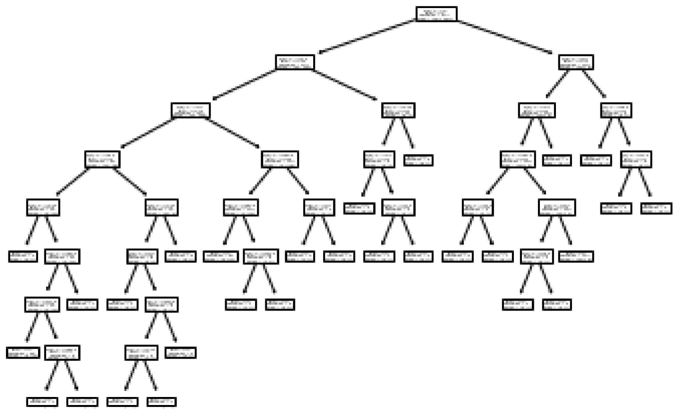
```
accuracy score:  0.8589743589743589
precision score:  0.7297297297297297
recall score:  0.9642857142857143
f1 score:  0.8307692307692307

 Metrics using the classification report

              precision    recall  f1-score   support

           0       0.96      0.90      0.93        50
           1       0.84      0.93      0.88        28

    accuracy                           0.91        78
   macro avg       0.90      0.91      0.90        78
weighted avg       0.91      0.91      0.91        78

 Confusion_matrix results

[[45  5]
 [ 2 26]]
```

Plot the tree

In [425]:
```
# Plot the tree
plot_tree(DT)
```

```
Out[425]: [Text(0.6433823529411765, 0.9444444444444444, 'X[0] <= 2.5\ngini = 0.5\nsampl
es = 311\nvalue = [153, 158]'),
 Text(0.4338235294117647, 0.8333333333333334, 'X[2] <= 101.0\ngini = 0.239\ns
amples = 173\nvalue = [24, 149]'),
 Text(0.27941176470588236, 0.7222222222222222, 'X[5] <= 75.5\ngini = 0.179\ns
amples = 161\nvalue = [16, 145]'),
 Text(0.14705882352941177, 0.6111111111111112, 'X[1] <= 119.5\ngini = 0.362\n
samples = 59\nvalue = [14, 45]'),
 Text(0.058823529411764705, 0.5, 'X[4] <= 13.75\ngini = 0.159\nsamples = 46\n
value = [4, 42]'),
 Text(0.029411764705882353, 0.3888888888888889, 'gini = 0.0\nsamples = 2\nval
ue = [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\nv
alue = [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\nva
lue = [1, 0]'),
 Text(0.11764705882352941, 0.05555555555555555, 'gini = 0.0\nsamples = 4\nval
ue = [0, 4]'),
 Text(0.11764705882352941, 0.2777777777777778, 'gini = 0.0\nsamples = 1\nvalu
e = [1, 0]'),
 Text(0.23529411764705882, 0.5, 'X[4] <= 17.75\ngini = 0.355\nsamples = 13\nv
alue = [10, 3]'),
 Text(0.20588235294117646, 0.3888888888888889, 'X[2] <= 81.5\ngini = 0.469\ns
amples = 8\nvalue = [5, 3]'),
 Text(0.17647058823529413, 0.2777777777777778, 'gini = 0.0\nsamples = 2\nvalu
e = [0, 2]'),
 Text(0.23529411764705882, 0.2777777777777778, 'X[3] <= 2329.5\ngini = 0.278
\nsamples = 6\nvalue = [5, 1]'),
 Text(0.20588235294117646, 0.16666666666666666, 'X[4] <= 14.75\ngini = 0.5\ns
amples = 2\nvalue = [1, 1]'),
 Text(0.17647058823529413, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nval
ue = [1, 0]'),
 Text(0.23529411764705882, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nval
ue = [0, 1]'),
 Text(0.2647058823529412, 0.16666666666666666, 'gini = 0.0\nsamples = 4\nvalu
e = [4, 0]'),
 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\n
samples = 102\nvalue = [2, 100]'),
 Text(0.35294117647058826, 0.5, 'X[3] <= 2880.0\ngini = 0.02\nsamples = 100\n
value = [1, 99]'),
 Text(0.3235294117647059, 0.3888888888888889, 'gini = 0.0\nsamples = 94\nvalu
e = [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\nvalu
e = [1, 0]'),
 Text(0.4117647058823529, 0.2777777777777778, 'gini = 0.0\nsamples = 5\nvalue
= [0, 5]'),
 Text(0.47058823529411764, 0.5, 'X[5] <= 77.5\ngini = 0.5\nsamples = 2\nvalue
```

```
= [1, 1]'),
 Text(0.4411764705882353, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nvalue
= [1, 0]'),
 Text(0.5, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.5882352941176471, 0.7222222222222222, 'X[4] <= 14.45\ngini = 0.444\ns
amples = 12\nvalue = [8, 4]'),
 Text(0.5588235294117647, 0.6111111111111112, 'X[5] <= 76.0\ngini = 0.444\nsa
mples = 6\nvalue = [2, 4]'),
 Text(0.5294117647058824, 0.5, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.5882352941176471, 0.5, 'X[2] <= 107.5\ngini = 0.444\nsamples = 3\nval
ue = [2, 1]'),
 Text(0.5588235294117647, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nvalue
= [0, 1]'),
 Text(0.6176470588235294, 0.3888888888888889, 'gini = 0.0\nsamples = 2\nvalue
= [2, 0]'),
 Text(0.6176470588235294, 0.6111111111111112, 'gini = 0.0\nsamples = 6\nvalue
= [6, 0]'),
 Text(0.8529411764705882, 0.8333333333333334, 'X[5] <= 79.5\ngini = 0.122\nsa
mples = 138\nvalue = [129, 9]'),
 Text(0.7941176470588235, 0.7222222222222222, 'X[4] <= 21.6\ngini = 0.045\nsa
mples = 129\nvalue = [126, 3]'),
 Text(0.7647058823529411, 0.6111111111111112, 'X[3] <= 2737.0\ngini = 0.031\n
samples = 128\nvalue = [126, 2]'),
 Text(0.7058823529411765, 0.5, 'X[2] <= 111.0\ngini = 0.444\nsamples = 3\nval
ue = [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\nva
lue = [124, 1]'),
 Text(0.7941176470588235, 0.3888888888888889, 'X[1] <= 225.0\ngini = 0.375\ns
amples = 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\nval
ue = [121, 0]'),
 Text(0.8235294117647058, 0.6111111111111112, 'gini = 0.0\nsamples = 1\nvalue
= [0, 1]'),
 Text(0.9117647058823529, 0.7222222222222222, 'X[1] <= 196.5\ngini = 0.444\ns
amples = 9\nvalue = [3, 6]'),
 Text(0.8823529411764706, 0.6111111111111112, 'gini = 0.0\nsamples = 4\nvalue
= [0, 4]'),
 Text(0.9411764705882353, 0.6111111111111112, 'X[1] <= 247.0\ngini = 0.48\nsa
mples = 5\nvalue = [3, 2]'),
 Text(0.9117647058823529, 0.5, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.9705882352941176, 0.5, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]')]
```



10. Neural Networks

Neural Network Classification (CNN)

Logistic regression as a base line

In [426]:
```
# Train the algorithm
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score
clf = LogisticRegression(max_iter=500)
clf.fit(X_train, y_train)
```

Out[426]:
```
LogisticRegression(max_iter=500)
```

In [427]:
```
# Test and evaluate
pred = clf.predict(X_test)
print('accuracy = ', accuracy_score(y_test, pred))
print("\n Confusion_matrix results\n")
print(confusion_matrix(y_test, pred))
```

```
accuracy =  0.8589743589743589

 Confusion_matrix results

[[40 10]
 [ 1 27]]
```

Compare to Neural Network classification

(CNN_1)

using the hidden layers (5,2)

In [428]:
```
# Scaling the data, it keeps whatever skew the data has.
from sklearn import preprocessing

# Sklearn scalar is fit to the training data only, then applied to both train
 and test.
scaler = preprocessing.StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [429]:
```
# Train neural network
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report,confusion_matrix
clf = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5, 2), max_iter=500, r
andom_state=1234)
clf.fit(X_train_scaled, y_train)
```

Out[429]:
```
MLPClassifier(hidden_layer_sizes=(5, 2), max_iter=500, random_state=1234,
              solver='lbfgs')
```

In [430]:
```
# Test and evaluate
pred = clf.predict(X_test_scaled)

# Output results
print('accuracy = ', accuracy_score(y_test, pred))
# Print the classification report metrics
print("\n Metrics using the classification report\n")
print(classification_report(y_test, pred))
print(" Confusion_matrix results\n")
print(confusion_matrix(y_test, pred))
```

```
accuracy =  0.8846153846153846

 Metrics using the classification report

              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

 Confusion_matrix results

[[44  6]
 [ 3 25]]
```

Compare to Neural Network classification

(CNN_2)

using only 3 nodes and a different setting, a different solver

In [431]:
```
# Train a second neural network
clf = MLPClassifier(solver='sgd', hidden_layer_sizes=(3,), max_iter=1500, rand
om_state=1234)
clf.fit(X_train_scaled, y_train)
```

Out[431]:
```
MLPClassifier(hidden_layer_sizes=(3,), max_iter=1500, random_state=1234,
              solver='sgd')
```

In [432]:
```
# Test and evaluate
pred = clf.predict(X_test_scaled)

# Output results
print('accuracy = ', accuracy_score(y_test, pred))
# Print the classification report metrics
print("\n Metrics using the classification report\n")
print(classification_report(y_test, pred))
print(" Confusion_matrix results\n")
confusion_matrix(y_test, pred)
```

```
accuracy =  0.8846153846153846

 Metrics using the classification report

              precision    recall  f1-score   support

           0       0.98      0.84      0.90        50
           1       0.77      0.96      0.86        28

    accuracy                           0.88        78
   macro avg       0.87      0.90      0.88        78
weighted avg       0.90      0.88      0.89        78

 Confusion_matrix results

```

Out[432]:
```
array([[42,  8],
       [ 1, 27]])
```

Comparison of the above two models:

The two models outperformed the simple logistic regression with accuracy 85 %, both have 88 % accuracy.The reason that they performed equally could be overfitting. Complex models may overfit when using small data sets.

11. Analysis

a. Decision Tree, overall perfomed the best on the Auto data.

b. Compare accuracy, recall and precision metrics by class:

Accuracy: Decision Tree, with 91 % accuracy performed best than Logistic Regression, with 86 %, and both CNN, with 88 % .

Class 0

- Recall : Decision Tree (90 %) performed better than Logistic Regression (80 %)CNN_1 (88 %) and CNN_2 (84 %)
- Precision: Both Logistic Regression and CNN_2(98 %) performed better than Decision Tree(96 %) and CNN_1 (94 %)

Class 1

- Recall: Both Logistic Regression and CNN_2(96 %) performed better than Decision Tree (93 %) and CNN_1 (89 %)
- Precision: Decision Tree (84 %) performed better than Logistic Regression(73 %), CNN_2 (77 %), and CNN_1 (81 %).

c. Decision Tree performed better because of the nature of the data set, the automobile's attributes is complex, with a lot of overlap between cylinders, horsepower and acceleration that cannot be linearly modeled. Simple models couldn't learn complex mapping functions. Outliers in data could have skewed the logistic regression, thus fail to separate the classes as much as Decision Trees, with robust algorithm does.

d. Both R and sklearn have special features, I like compared to other IDE's I used to code, like running chunck by chunk. I liked sklearn more that the run time is fast, and easy to print into pdf.