Assignment 5: ML with sklearn

Name: Aarya Patil

Date: 04/05/23

In [ ]:
```
# Import libraries
import pandas as pd
import seaborn as sb
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.neural_network import MLPClassifier
```

1. Read the Auto data

In [ ]:
```
# Read the data
df = pd.read_csv('Auto.csv')

# Output the first few rows
print(df.head(n = 5).to_string(index = False))

# Output the dimensions
print('\nDimensions:', df.shape)
```

```
 mpg  cylinders  displacement  horsepower  weight  acceleration  year  orig
in                        name
18.0          8         307.0         130    3504          12.0  70.0
1 chevrolet chevelle malibu
15.0          8         350.0         165    3693          11.5  70.0
1         buick skylark 320
18.0          8         318.0         150    3436          11.0  70.0
1        plymouth satellite
16.0          8         304.0         150    3433          12.0  70.0
1             amc rebel sst
17.0          8         302.0         140    3449           NaN  70.0
1               ford torino

Dimensions: (392, 9)
```

2. Data exploration with code

In [ ]:
```
# Use describe() on the mpg, weight, and year columns
print(df[["mpg", "weight", "year"]].describe(include = "all"))
```

```
             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
```

Comments on the data exploration:

- Mpg -> Range: 37, Average: 23.445918
- Weight -> Range: 3527, Average: 2977.584184
- Year -> Range: 12, Average: 76.010256

3. Explore data types

In [ ]:
```
# Check the data types of all columns
print('ORIGINAL DATATYPES:')
print(df.dtypes)

# 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
print('\nCHANGED DATATYPES:')
print(df.dtypes)
```

```
ORIGINAL DATATYPES:
mpg             float64
cylinders         int64
displacement    float64
horsepower        int64
weight            int64
acceleration    float64
year            float64
origin            int64
name             object
dtype: object

CHANGED DATATYPES:
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 [ ]:

```
# Delete rows with NAs
df = df.dropna()

# Output the new dimensions
print('New dimensions:', df.shape)
```

```
New dimensions: (389, 9)
```

5. Modify columns

In [ ]:
```
# Make a new column, mpg_high (categorical)
high_or_low = []  # list to hold 0 or 1 values based on whether the mpg is h
i = 0  # counter variable

for item in df['mpg']:  # assign whether the mpg is high(1) or low(0) by com
    if item > df['mpg'].mean():
        high_or_low.insert(i, 1)
    else:
        high_or_low.insert(i, 0)
    i = i+1

df['mpg_high'] = high_or_low  # add the new column and assign the values to
df['mpg_high'] = df['mpg_high'].astype('category')  # change the data type t
# df.dtypes  # debug statement to make sure the type is changed

# Delete the mpg and name columns
df = df.drop(columns=['mpg', 'name'])

# Output the first few rows of the modified data frame
df.head()
```

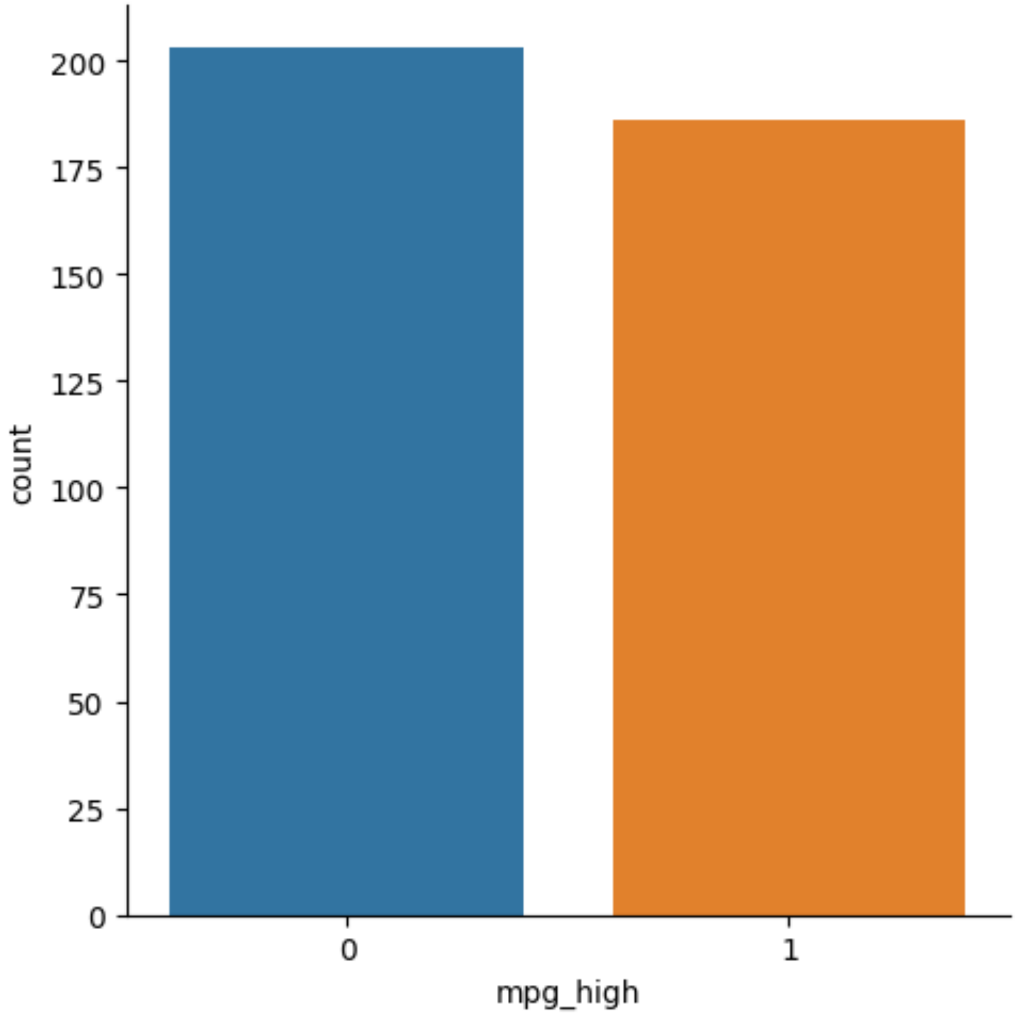
Out[ ]:

	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

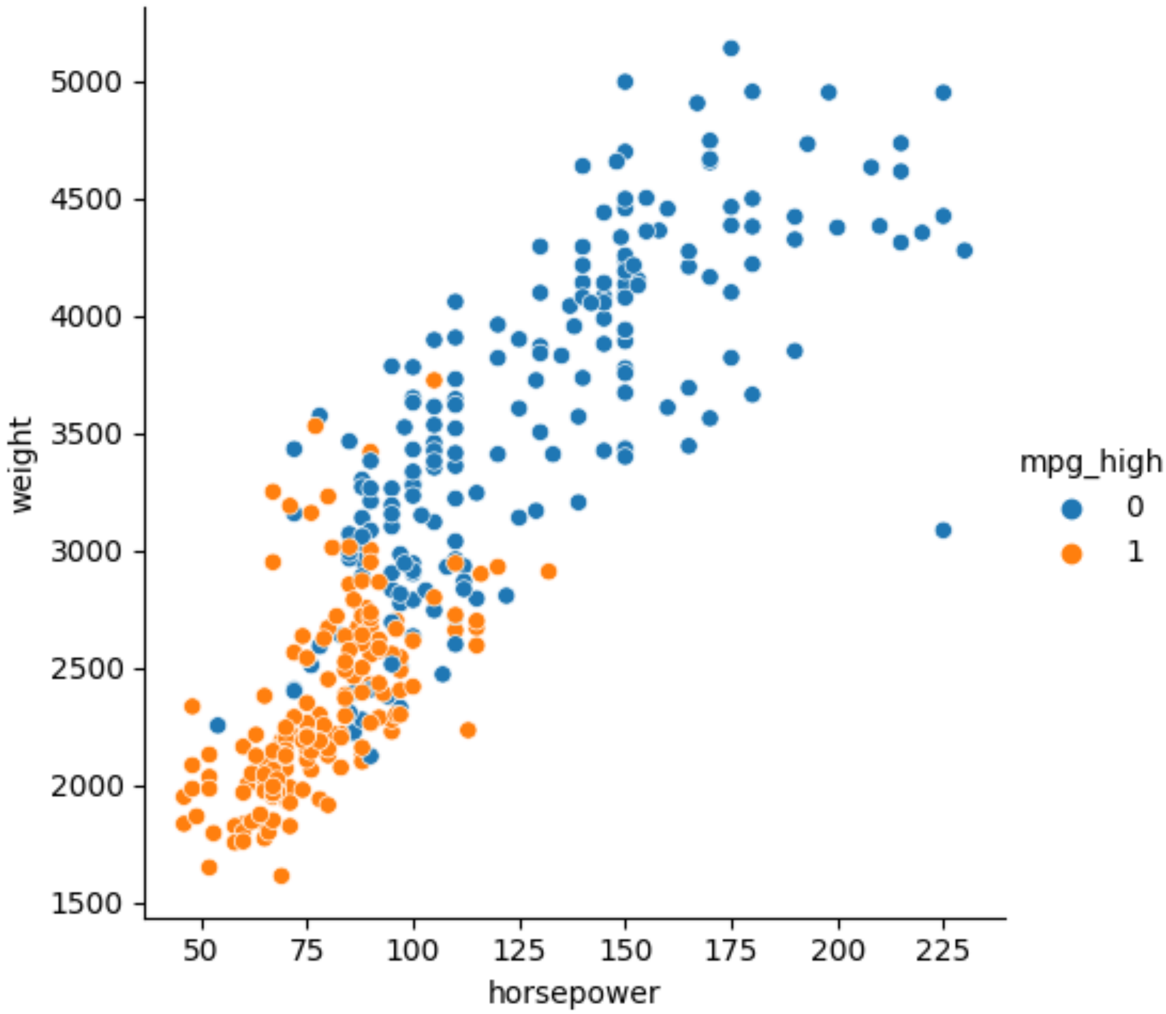
In [ ]:
```
# Seaborn catplot on the mpg_high column
sb.catplot(data = df, x = 'mpg_high', kind = 'count')
```

Out[ ]:
```
<seaborn.axisgrid.FacetGrid at 0x29d9de890>
```



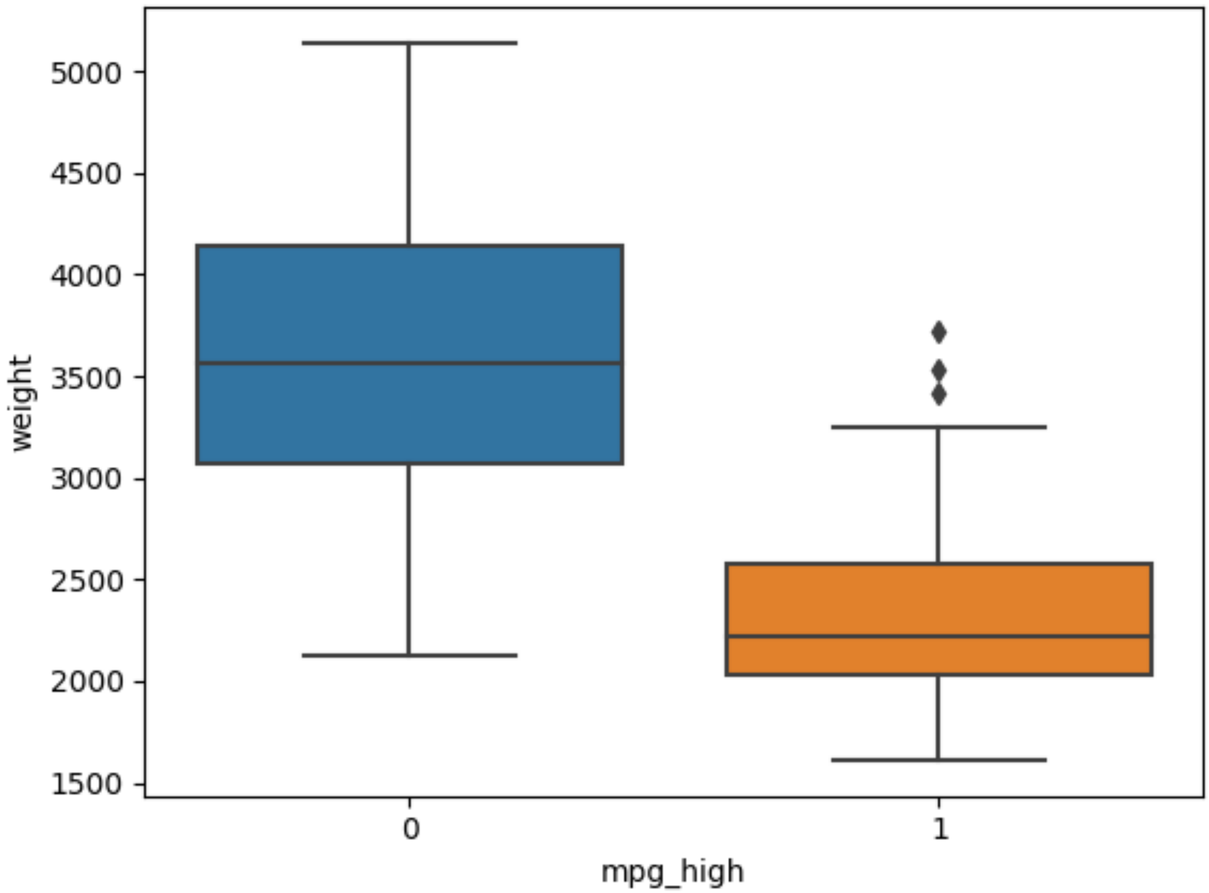
In [ ]:
```
# Seaborn relplot with horsepower on the x axis, weight on the y axis, and s
sb.relplot(data = df, x ='horsepower', y = 'weight', hue = 'mpg_high')
```

Out[ ]: <seaborn.axisgrid.FacetGrid at 0x29d98ab00>



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

Out[ ]: <Axes: xlabel='mpg_high', ylabel='weight'>



What I learned about the data from each graph:

- Catplot: I got to see how many cars have a high mpg vs. a low mpg.
- Relplot: I learned that as the weight of the car increases, so does the horsepower. The relationship they have is linear. Also cars that weigh less and have a lower horsepower tend to have a higher mpg.
- Boxplot: The second part that I learned from the replot was confirmed in this graph. Cars that weight less tend to have a higher mpg. I also got the see the range and average weight the cars with both high and low mpgs fall in.

7. Train/test split

In [ ]:
```
# Train/test X data frames consists of all remaining columns except mpg_high
X = df.loc[:, ['cylinders', 'displacement', 'horsepower', 'weight', 'acceler
y = df['mpg_high']

# Split the training and testing data 80/20
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 0.8,

# Output the dimensions of train and test data
print('X train size:', X_train.shape)
print('X test size:', X_test.shape)
print('y train size:', y_train.shape)
print('y test size:', y_test.shape)
```

```
X train size: (311, 7)
X test size: (78, 7)
y train size: (311,)
y test size: (78,)
```

8. Logistic Regression

In [ ]:
```
# Train a logistic regression model using solver lbfgs
logReg = LogisticRegression(solver = 'lbfgs', max_iter = 200)
logReg.fit(X_train, y_train)

# Test and evaluate
pred = logReg.predict(X_test)

# Print metrics using the classification report
print(classification_report(y_test, pred))
```

```
              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
```

9. Decision Tree

In [ ]:
```
# Train a decision tree
decTree = DecisionTreeClassifier()
decTree.fit(X_train, y_train)

# Test and evaluate
pred2 = decTree.predict(X_test)

# Print metrics using the classification report
print(classification_report(y_test, pred2))

# Plot the tree
print(tree.plot_tree(decTree))
```

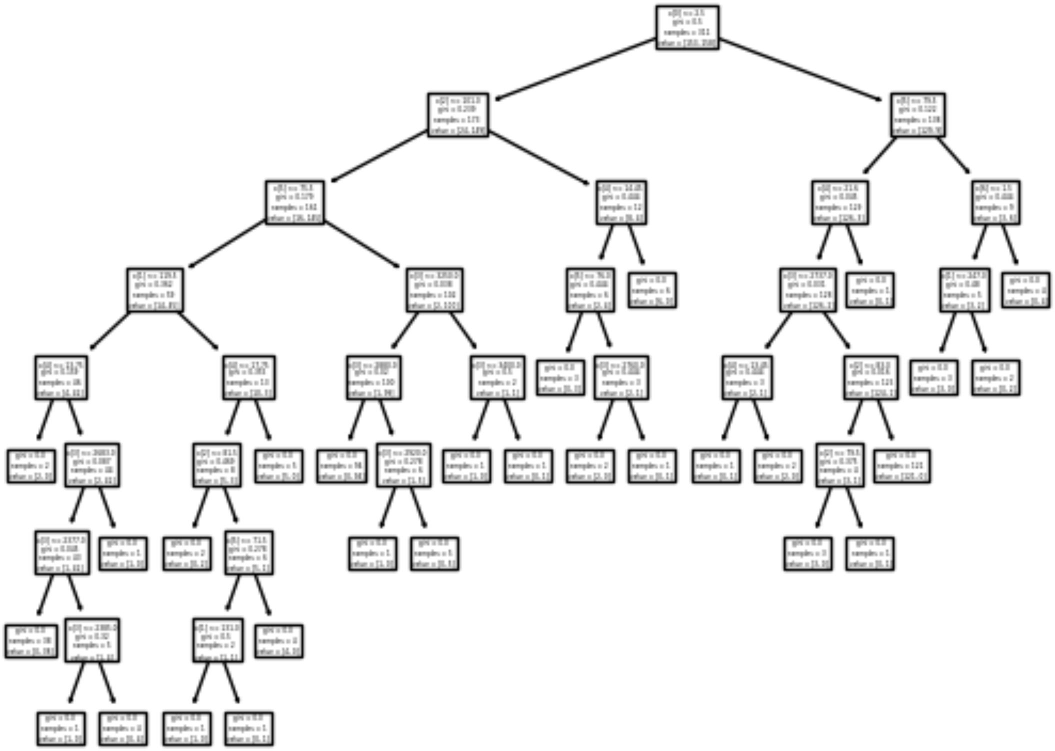
```
              precision    recall  f1-score   support

           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
```

```
[Text(0.6507352941176471, 0.9444444444444444, 'x[0] <= 2.5\ngini = 0.5\nsam
ples = 311\nvalue = [153, 158]'), Text(0.4338235294117647, 0.83333333333333
34, 'x[2] <= 101.0\ngini = 0.239\nsamples = 173\nvalue = [24, 149]'), Text(
0.27941176470588236, 0.7222222222222222, 'x[5] <= 75.5\ngini = 0.179\nsampl
es = 161\nvalue = [16, 145]'), Text(0.14705882352941177, 0.6111111111111112
, 'x[1] <= 119.5\ngini = 0.362\nsamples = 59\nvalue = [14, 45]'), Text(0.05
8823529411764705, 0.5, 'x[4] <= 13.75\ngini = 0.159\nsamples = 46\nvalue =
[4, 42]'), Text(0.029411764705882353, 0.3888888888888889, 'gini = 0.0\nsamp
les = 2\nvalue = [2, 0]'), Text(0.08823529411764706, 0.3888888888888889, 'x
[3] <= 2683.0\ngini = 0.087\nsamples = 44\nvalue = [2, 42]'), Text(0.058823
529411764705, 0.2777777777777778, 'x[3] <= 2377.0\ngini = 0.045\nsamples =
43\nvalue = [1, 42]'), Text(0.029411764705882353, 0.16666666666666666, 'gin
i = 0.0\nsamples = 38\nvalue = [0, 38]'), Text(0.08823529411764706, 0.16666
666666666666, 'x[3] <= 2385.0\ngini = 0.32\nsamples = 5\nvalue = [1, 4]'),
Text(0.058823529411764705, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nv
alue = [1, 0]'), Text(0.11764705882352941, 0.05555555555555555, 'gini = 0.0
\nsamples = 4\nvalue = [0, 4]'), Text(0.11764705882352941, 0.27777777777777
78, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'), Text(0.23529411764705882, 0
.5, 'x[4] <= 17.75\ngini = 0.355\nsamples = 13\nvalue = [10, 3]'), Text(0.2
0588235294117646, 0.3888888888888889, 'x[2] <= 81.5\ngini = 0.469\nsamples
= 8\nvalue = [5, 3]'), Text(0.17647058823529413, 0.2777777777777778, 'gini
= 0.0\nsamples = 2\nvalue = [0, 2]'), Text(0.23529411764705882, 0.277777777
7777778, 'x[5] <= 71.5\ngini = 0.278\nsamples = 6\nvalue = [5, 1]'), Text(0
.20588235294117646, 0.16666666666666666, 'x[1] <= 131.0\ngini = 0.5\nsample
s = 2\nvalue = [1, 1]'), Text(0.17647058823529413, 0.05555555555555555, 'gi
ni = 0.0\nsamples = 1\nvalue = [1, 0]'), Text(0.23529411764705882, 0.055555
55555555555, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'), Text(0.26470588235
29412, 0.16666666666666666, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]'), Tex
```

```
t(0.2647058823529412, 0.3888888888888889, 'gini = 0.0\nsamples = 5\nvalue =
[5, 0]'), Text(0.4117647058823529, 0.6111111111111112, 'x[3] <= 3250.0\ngin
i = 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(0.32
35294117647059, 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, 0.2777777
777777778, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'), Text(0.4117647058823
529, 0.2777777777777778, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'), Text(0
.47058823529411764, 0.5, 'x[3] <= 3400.0\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\nsamples = 12\nvalue = [8, 4]'), Text(0.558823529411764
7, 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, 0.388888888888
8889, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'), Text(0.6176470588235294,
0.6111111111111112, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]'), Text(0.8676
470588235294, 0.8333333333333334, 'x[5] <= 79.5\ngini = 0.122\nsamples = 13
8\nvalue = [129, 9]'), Text(0.7941176470588235, 0.7222222222222222, 'x[4] <
= 21.6\ngini = 0.045\nsamples = 129\nvalue = [126, 3]'), Text(0.76470588235
29411, 0.6111111111111112, 'x[3] <= 2737.0\ngini = 0.031\nsamples = 128\nva
lue = [126, 2]'), Text(0.7058823529411765, 0.5, 'x[4] <= 13.45\ngini = 0.44
4\nsamples = 3\nvalue = [2, 1]'), Text(0.6764705882352942, 0.38888888888888
89, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'), Text(0.7352941176470589, 0.
3888888888888889, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'), Text(0.823529
4117647058, 0.5, 'x[2] <= 83.0\ngini = 0.016\nsamples = 125\nvalue = [124,
1]'), Text(0.7941176470588235, 0.3888888888888889, 'x[2] <= 79.5\ngini = 0.
375\nsamples = 4\nvalue = [3, 1]'), Text(0.7647058823529411, 0.277777777777
7778, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'), Text(0.8235294117647058,
0.2777777777777778, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'), Text(0.8529
411764705882, 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\nva
lue = [0, 4]')]
```



10. Neural Network

In [ ]:
```
# Train first neural network
neuralNet = MLPClassifier(hidden_layer_sizes = (7, 6, 5, 4, 3, 2, 1),random_
neuralNet.fit(X_train, y_train)

# Test and evaluate
pred3 = neuralNet.predict(X_test)

# Print metrics using the classification report
print(classification_report(y_test, pred3))

# Train second neural network
neuralNet2 = MLPClassifier(hidden_layer_sizes = (6, 3), random_state = 1234,
neuralNet2.fit(X_train, y_train)

# Test and evaluate
pred4 = neuralNet2.predict(X_test)

# Print metrics using the classification report
print(classification_report(y_test, pred4))
```

```
              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

              precision    recall  f1-score   support

           0       0.88      0.86      0.87        50
           1       0.76      0.79      0.77        28

    accuracy                           0.83        78
   macro avg       0.82      0.82      0.82        78
weighted avg       0.83      0.83      0.83        78
```

Model Comparison

- My first neural network model did much better than my second. I think that the overfitting in the first model somehow worked and gave me great precision. The accuracy isn't great for either model but it isn't terrible. Overall, the first model did better or equal in every category.

11. Analysis

My better neural network model and the logistic regression model both performed equally as well. The decision tree performed worse only in precision for 0 and recall for 0 but out-performed both of the other algorithms in all the other categories.

The accuracy for logistic regression and the neural network was 0.87 and for the decision tree it was 0.90. The recall for logistic regression and the neural network was 0.82(for 0) and 0.96(for 1) and for the decision tree it was 0.90. The precision for logistic regression and the neural network was 0.98(for 0) and 0.75(for 1) and for the decision tree it was 0.92(for 0) and 0.86(for 1).

I think that the data given to us was well suited for the decision tree. Usually I've noticed that logistic regression does really well, followed by neural network. I also think that maybe the network topology that I picked for the neural network models definitely skewed the data and caused it to return values that aren't as accurate. If I was to analyze this data again I would try to go about doing it differently so that I could get accurate results in all the algorithms.

I personally enjoy both R and sklearn pretty equally. I prefer R at times since many of the functions are built in but it's not too difficult in sklearn either. There are also things I prefer about sklearn though, such as the visual appearance, and the ease of using it. Overall, I would say they are fairy matched but if I had to choose only one to continue using, it would definitely be sklearn.