Machine Learning with SKLearn

Yazan Abughazaleh

Reading Data with Pandas

The first step is to read in the Auto.csv file into a Pandas data frame.

In [ ]:
```
import pandas as pd
df = pd.read_csv('Auto.csv')
print("Auto data: ")
df.head()
```

```
Auto data:
```

Out[ ]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	8	307.0	130	3504	12.0	70.0	1	chevrolet chevelle malibu
1	15.0	8	350.0	165	3693	11.5	70.0	1	buick skylark 320
2	18.0	8	318.0	150	3436	11.0	70.0	1	plymouth satellite
3	16.0	8	304.0	150	3433	12.0	70.0	1	amc rebel sst
4	17.0	8	302.0	140	3449	NaN	70.0	1	ford torino

Above we can see the first 5 rows of our data frame. Next we would like to check the dimensionality of our data.

In [ ]:
```
print("DF dimensions: ",df.shape)
```

```
DF dimensions:  (392, 9)
```

The dimensions of the data frame indicate that there ar 392 instances with 9 attributes describing the instance.

Data Exploration

The next step to perform is to learn about our data using the describe function.

In [ ]:
```
print("Describe mpg, weight, and year: \n", df.loc[:,['mpg','weight','year']]
```

```
Describe mpg, weight, and year:
              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
```

Average for mpg: 23.446

Range for mpg: 37.6

Average for weight: 2977.584

Range for weight: 3527

Average for year: 76.010

Range for year: 12.0

Exploring Data Types

We now will explore the different data types used in the data frame. The first step is to check the data types of each column.

In [ ]:
```
df.dtypes
```

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

We can see the data types of each column displayed above. Now we would like to convert the cylinders and origin columns to factors.

In [ ]:
```
df.cylinders = df.cylinders.astype('category').cat.codes
df.origin = df.origin.astype('category')
df.dtypes
```

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

We can see above that the data types were successfully converted. The next step is to deal with NA values.

Dealing with NAs

We would like to ensure that our data has no NAs for training. We first check how many rows have NAs. If the count is low, we can safely remove the rows.

In [ ]:
```
print("NAs:\n",df.isnull().sum())
df = df.dropna()
print("DF dimensions: ",df.shape)
```

```
NAs:
 mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    1
year            2
origin          0
name            0
dtype: int64
DF dimensions:  (389, 9)
```

The new dimensions of the data indicate that three NA values were removed. We can now continue on to the next step.

Modifying Data

We are going to add a new column called "mpg_high" which is a categorical column that is based on the average mpg. Then, the "name" and "mpg" columns will be removed.

In [ ]:
```
df['mpg_high'] = df['mpg'].mean()
df['mpg_high'] = pd.cut(df['mpg'],[0,df['mpg'].mean(),float('Inf')] ,labels=
df = df.drop(columns=['mpg','name'])
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

Above, we can see that the data frame no longer has the "mpg" and "name" columns. Now we can try visual data exploration with Seaborn.

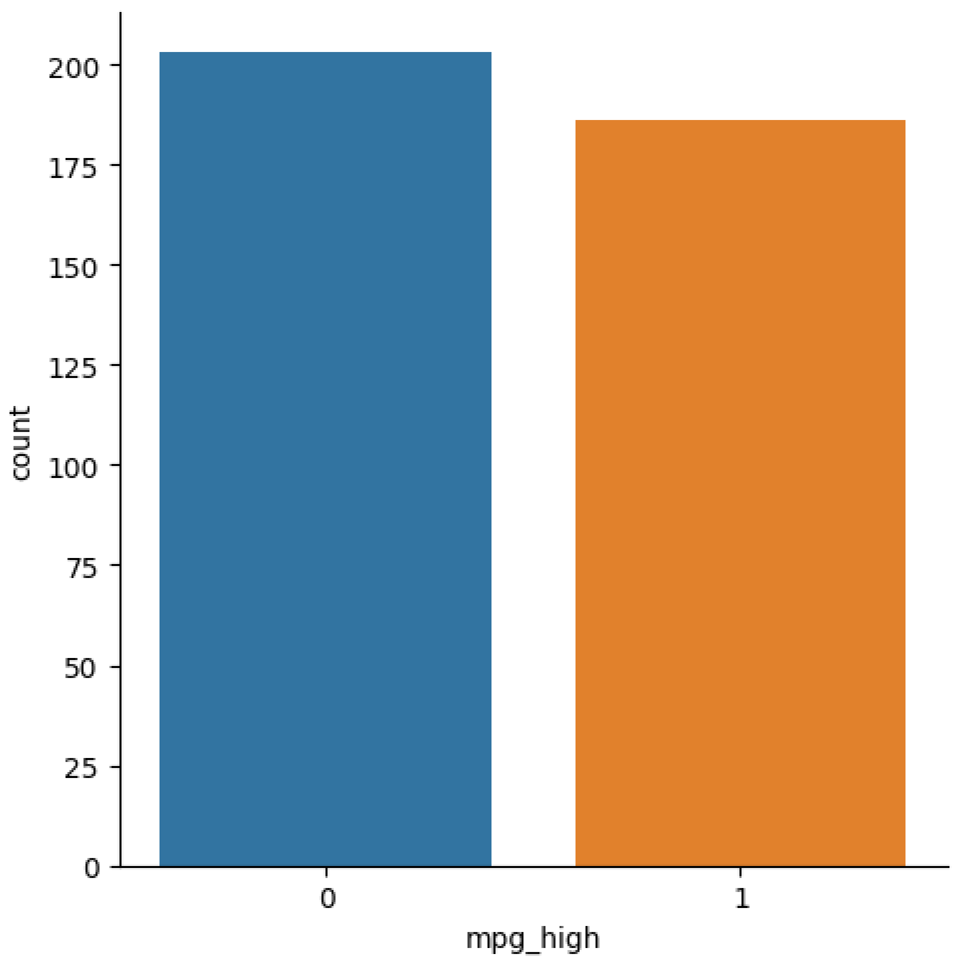
Data Exploration With Graphs

To get started with data exploration using graphs, we first need to import the Seaborn package. We will then create a categorical plot of the "mpg_high" column.

In [ ]:

```
import seaborn as sb
sb.catplot(x="mpg_high",kind="count",data=df)
```

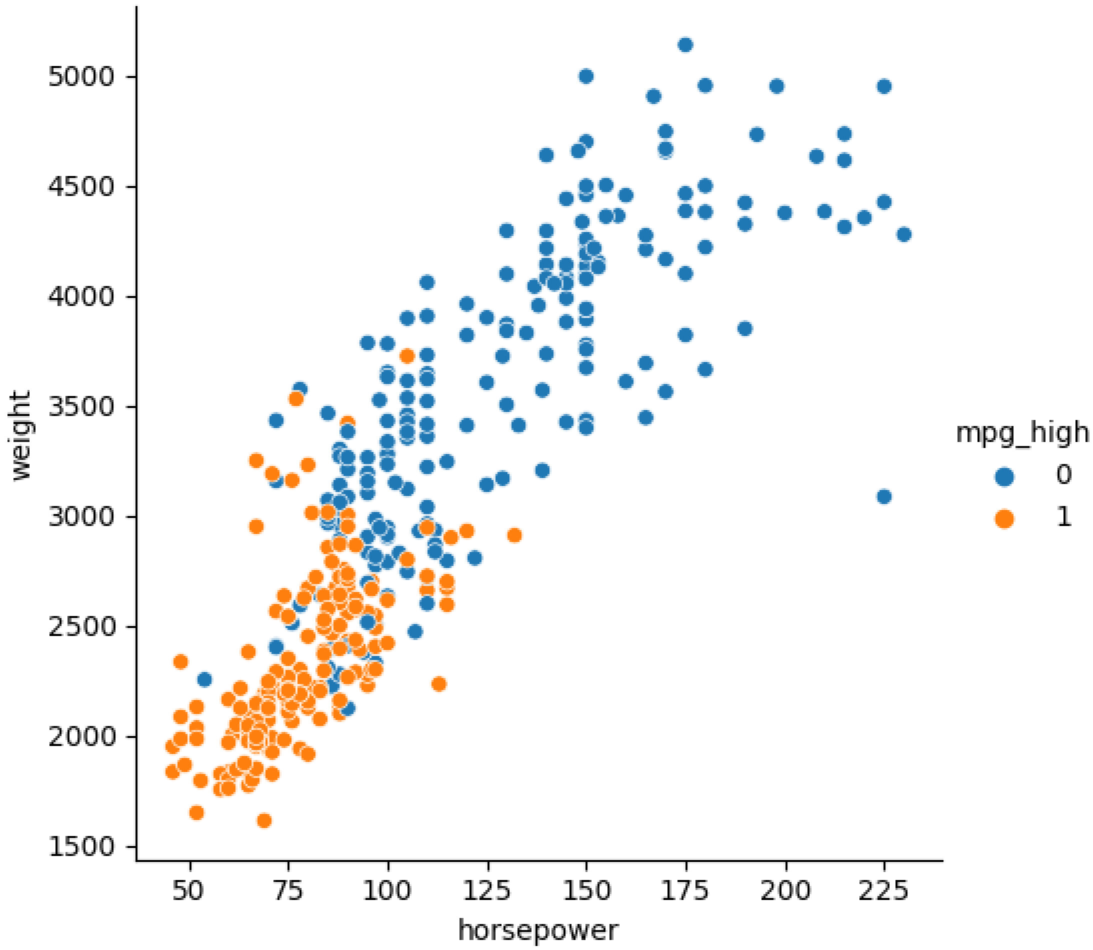
Out[ ]: <seaborn.axisgrid.FacetGrid at 0x122f95910>



Next is a relational plot between between horsepower and weight.

In [ ]:
```
sb.relplot(x="horsepower",y="weight",data=df,hue=df.mpg_high)
```

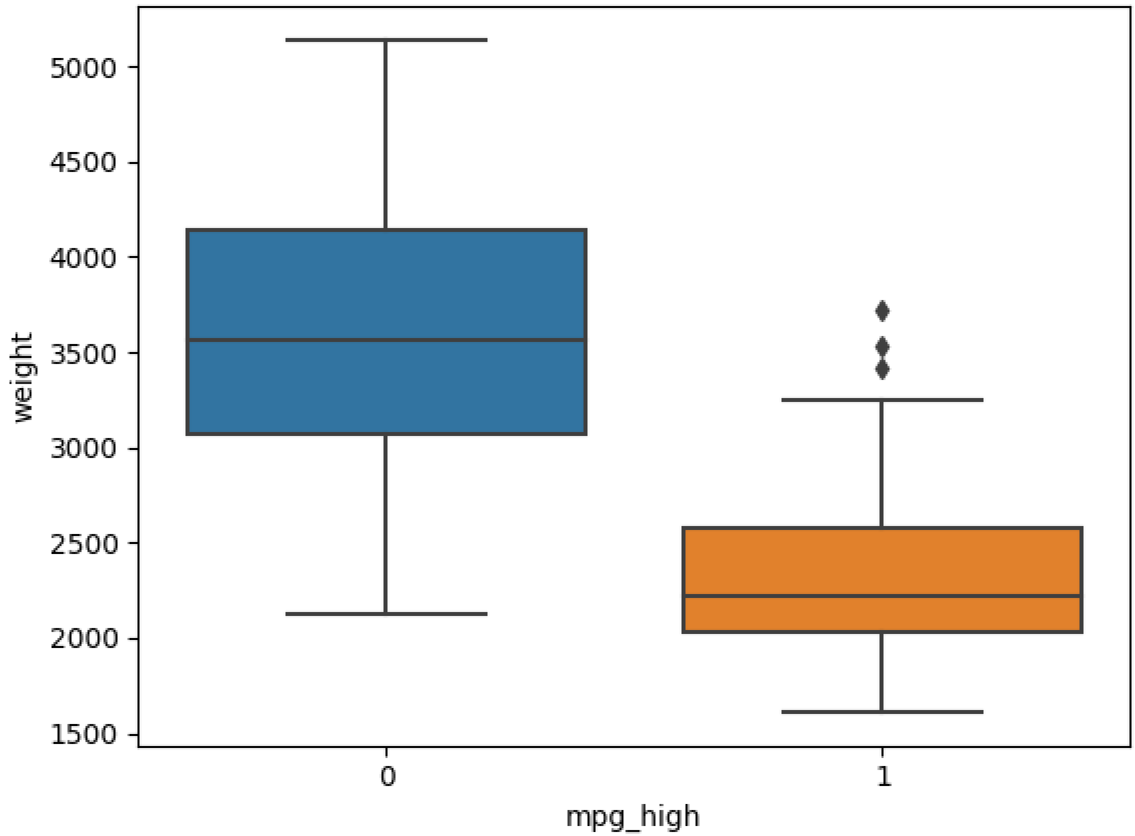
Out[ ]: <seaborn.axisgrid.FacetGrid at 0x122e87650>



The next plot is a box plot of mpg high and weight.

In [ ]:
```
sb.boxplot(x="mpg_high",y="weight",data=df)
```

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



The first plot shows how many vehicles are in the low gas milage category and high gas mileage category. The second plot shows the relationship between the horsepower and weight of each vehicle, and the color is indicative of whether the gas milage is high or low. The trend shows that high gas milage vehicles typically have lower horsepower while low gas milage cars have higher horsepower. The last plot depicts the weights of the vehicls and their corresponding mpg classification.

Train Test Data Split

The Auto data will now be split using an 80/20 split to create train and test data sets.

In [ ]:
```
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, random_state=1234)
print("train size:", X_train.shape)
print("test size:", X_test.shape)
```

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

Logistic Regression

We will now use logistic regression to predict the mpg_high column.

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
clf = LogisticRegression(max_iter=150)
clf.fit(X_train,y_train)
clf.score(X_train,y_train)
pred = clf.predict(X_test)
print(classification_report(y_test, pred))
print(confusion_matrix(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.86      0.91        50
           1       0.79      0.96      0.87        28

    accuracy                           0.90        78
   macro avg       0.89      0.91      0.89        78
weighted avg       0.91      0.90      0.90        78

[[43  7]
 [ 1 27]]
```

We can see the accuracy printed above is at 90% and we also have a printout of the confusion matrix showing how the type of classifications.

Decision Trees

We will now use decision trees to perform the classification to compare the results with logistic regression.

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
dct = DecisionTreeClassifier()
dct.fit(X_train, y_train)
pred = dct.predict(X_test)
print(classification_report(y_test, pred))
tree.plot_tree(dct)
```

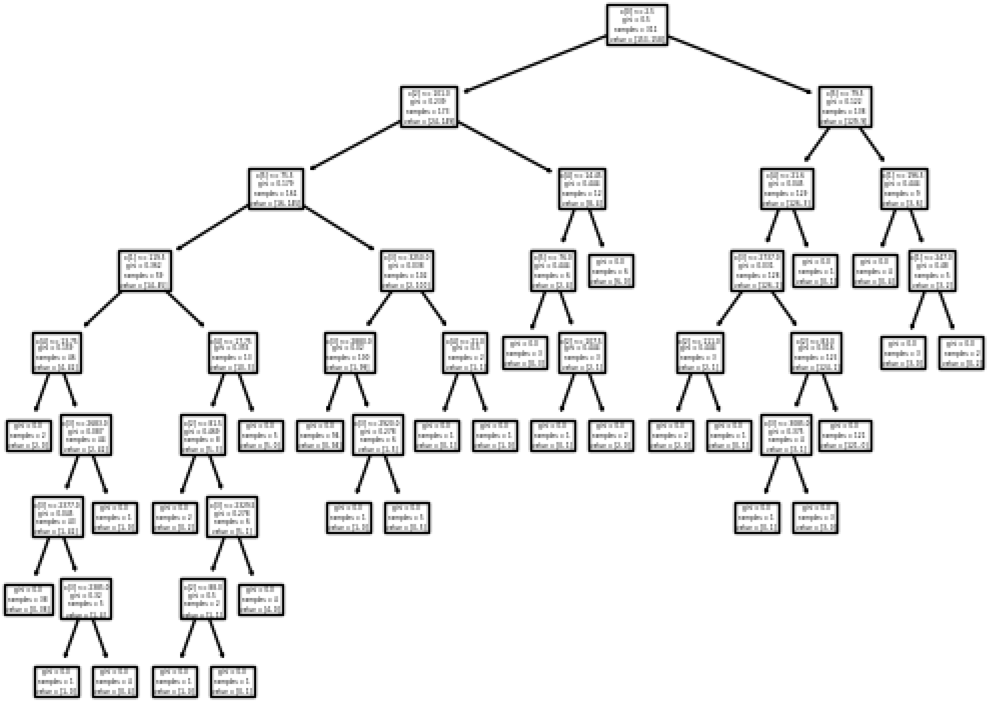
```
              precision    recall  f1-score   support

           0       0.96      0.92      0.94        50
           1       0.87      0.93      0.90        28

    accuracy                           0.92        78
   macro avg       0.91      0.92      0.92        78
weighted avg       0.93      0.92      0.92        78
```

```
Out[ ]: [Text(0.6433823529411765, 0.9444444444444444, 'x[0] <= 2.5\ngini = 0.5\nsam
        ples = 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\nv
        alue = [2, 0]'),
         Text(0.08823529411764706, 0.3888888888888889, 'x[3] <= 2683.0\ngini = 0.08
        7\nsamples = 44\nvalue = [2, 42]'),
         Text(0.058823529411764705, 0.2777777777777778, 'x[3] <= 2377.0\ngini = 0.0
        45\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 = 0.3
        2\nsamples = 5\nvalue = [1, 4]'),
         Text(0.058823529411764705, 0.05555555555555555, 'gini = 0.0\nsamples = 1\n
        value = [1, 0]'),
         Text(0.11764705882352941, 0.05555555555555555, 'gini = 0.0\nsamples = 4\nv
        alue = [0, 4]'),
         Text(0.11764705882352941, 0.2777777777777778, 'gini = 0.0\nsamples = 1\nva
        lue = [1, 0]'),
         Text(0.23529411764705882, 0.5, 'x[4] <= 17.75\ngini = 0.355\nsamples = 13
        \nvalue = [10, 3]'),
         Text(0.20588235294117646, 0.3888888888888889, 'x[2] <= 81.5\ngini = 0.469
        \nsamples = 8\nvalue = [5, 3]'),
         Text(0.17647058823529413, 0.2777777777777778, 'gini = 0.0\nsamples = 2\nva
        lue = [0, 2]'),
         Text(0.23529411764705882, 0.2777777777777778, 'x[3] <= 2329.5\ngini = 0.27
        8\nsamples = 6\nvalue = [5, 1]'),
         Text(0.20588235294117646, 0.16666666666666666, 'x[2] <= 88.0\ngini = 0.5\n
        samples = 2\nvalue = [1, 1]'),
         Text(0.17647058823529413, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nv
        alue = [1, 0]'),
         Text(0.23529411764705882, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nv
        alue = [0, 1]'),
         Text(0.2647058823529412, 0.16666666666666666, 'gini = 0.0\nsamples = 4\nva
        lue = [4, 0]'),
         Text(0.2647058823529412, 0.3888888888888889, 'gini = 0.0\nsamples = 5\nval
        ue = [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\nva
        lue = [0, 94]'),
         Text(0.38235294117647056, 0.3888888888888889, 'x[3] <= 2920.0\ngini = 0.27
        8\nsamples = 6\nvalue = [1, 5]'),
         Text(0.35294117647058826, 0.2777777777777778, 'gini = 0.0\nsamples = 1\nva
        lue = [1, 0]'),
         Text(0.4117647058823529, 0.2777777777777778, 'gini = 0.0\nsamples = 5\nval
        ue = [0, 5]'),
```

```
 Text(0.47058823529411764, 0.5, 'x[4] <= 21.0\ngini = 0.5\nsamples = 2\nval
ue = [1, 1]'),
 Text(0.4411764705882353, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nval
ue = [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\n
samples = 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\nv
alue = [2, 1]'),
 Text(0.5588235294117647, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nval
ue = [0, 1]'),
 Text(0.6176470588235294, 0.3888888888888889, 'gini = 0.0\nsamples = 2\nval
ue = [2, 0]'),
 Text(0.6176470588235294, 0.6111111111111112, 'gini = 0.0\nsamples = 6\nval
ue = [6, 0]'),
 Text(0.8529411764705882, 0.8333333333333334, 'x[5] <= 79.5\ngini = 0.122\n
samples = 138\nvalue = [129, 9]'),
 Text(0.7941176470588235, 0.7222222222222222, 'x[4] <= 21.6\ngini = 0.045\n
samples = 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[2] <= 111.0\ngini = 0.444\nsamples = 3\nv
alue = [2, 1]'),
 Text(0.6764705882352942, 0.3888888888888889, 'gini = 0.0\nsamples = 2\nval
ue = [2, 0]'),
 Text(0.7352941176470589, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nval
ue = [0, 1]'),
 Text(0.8235294117647058, 0.5, 'x[2] <= 83.0\ngini = 0.016\nsamples = 125\n
value = [124, 1]'),
 Text(0.7941176470588235, 0.3888888888888889, 'x[3] <= 3085.0\ngini = 0.375
\nsamples = 4\nvalue = [3, 1]'),
 Text(0.7647058823529411, 0.2777777777777778, 'gini = 0.0\nsamples = 1\nval
ue = [0, 1]'),
 Text(0.8235294117647058, 0.2777777777777778, 'gini = 0.0\nsamples = 3\nval
ue = [3, 0]'),
 Text(0.8529411764705882, 0.3888888888888889, 'gini = 0.0\nsamples = 121\nv
alue = [121, 0]'),
 Text(0.8235294117647058, 0.6111111111111112, 'gini = 0.0\nsamples = 1\nval
ue = [0, 1]'),
 Text(0.9117647058823529, 0.7222222222222222, 'x[1] <= 196.5\ngini = 0.444
\nsamples = 9\nvalue = [3, 6]'),
 Text(0.8823529411764706, 0.6111111111111112, 'gini = 0.0\nsamples = 4\nval
ue = [0, 4]'),
 Text(0.9411764705882353, 0.6111111111111112, 'x[1] <= 247.0\ngini = 0.48\n
samples = 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]')]
```



The decision tree yielded a slight improvement in accuracy up to 92% over logistic regression.

Neural Networks

The final algorithm we will look at in this notebook will be neural networks. We will attempt to perform classification with two different topologies of neural networks. Neural networks tend to work best with scaled data, so the first step we will take is scale our data to make it easier to work with.

In [ ]:
```
from sklearn import preprocessing
from sklearn.neural_network import MLPClassifier
scaler = preprocessing.StandardScaler().fit(X_train)
X_trainScale = scaler.transform(X_train)
X_testScale = scaler.transform(X_test)

nn1 = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(3, 2),
    max_iter=500, random_state=1234)
nn1.fit(X_trainScale,y_train)
pred = nn1.predict(X_test)
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.64      1.00      0.78        50
           1       0.00      0.00      0.00        28

    accuracy                           0.64        78
   macro avg       0.32      0.50      0.39        78
weighted avg       0.41      0.64      0.50        78
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/sklearn/base.py:432: UserWarning: X has feature names, but MLPClassifier was fitted without feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Above we can see the results of the neural network with a topology of 5 hidden nodes. The accuracy is at 90%. The quantity of nodes was picked using the 2/3 rule for hidden node count. Next we will try the same process with a different topology.

In [ ]:
```
nn2 = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(7, 5),
    max_iter=1000, random_state=1234)
nn2.fit(X_trainScale,y_train)
pred = nn2.predict(X_test)
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.64      1.00      0.78        50
           1       0.00      0.00      0.00        28

    accuracy                           0.64        78
   macro avg       0.32      0.50      0.39        78
weighted avg       0.41      0.64      0.50        78
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/base.py:432: UserWarning: X has feature names, but MLPClassifi
er was fitted without feature names
  warnings.warn(
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Preci
sion and F-score are ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Preci
sion and F-score are ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-pack
ages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Preci
sion and F-score are ill-defined and being set to 0.0 in labels with no pre
dicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Increasing the node count here did not boost the accuracy but instead boosted the precision of the model.

Analysis

When comparing a neural network, logistic regression, and a decision tree, the decision tree performed best on the Autos data set with the highest accuracy of 92%. Tied for the highest precision metric for the low gas milage category are the neural network and logistic regression models. The precision for the high milage category is best with the decision tree. The recall for the low gas milage is best with the decision tree and the high gas milage is best on the neural network and logistic regression models. The likely reason that the decision tree performed the best is because the data set is relatively small, and algorithms such as the neural network typically require lots of data to outperform simpler algorithms. Simpler algorithms tend to work better with small data sets, and this is the case with the decision tree. Lastly, the experience using Python and SKLearn has been much easier in my opinion than R. I found that the standardized formats for many of the models is easy to follow and did not require that I change up the data structure depending on the package. Python is also easier to use than the R for me because it is structured more like conventional programming languages.