sklearn-rcd180001

November 4, 2022

1 ML with SKLearn

Joshua Durana rcd180001

[ ]:
```
import pandas as pd
import sklearn as sk
import seaborn as sb
import numpy as np
```

1.1 Load Data

[ ]:
```
autodf = pd.read_csv('Data/Auto.csv')
print("Head\n", autodf.head())
print("Dimensions: ", autodf.shape)
```

```
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
Dimensions:  (392, 9)
```

1.2 Data Exploration

[ ]:
```
autodf.describe()
```

[ ]:
```
              mpg   cylinders  displacement  horsepower       weight  \
count  392.000000  392.000000    392.000000  392.000000   392.000000
mean    23.445918    5.471939    194.411990  104.469388  2977.584184
```

```
std      7.805007    1.705783    104.644004   38.491160   849.402560
min      9.000000    3.000000     68.000000   46.000000  1613.000000
25%     17.000000    4.000000    105.000000   75.000000  2225.250000
50%     22.750000    4.000000    151.000000   93.500000  2803.500000
75%     29.000000    8.000000    275.750000  126.000000  3614.750000
max     46.600000    8.000000    455.000000  230.000000  5140.000000

       acceleration        year      origin
count    391.000000  390.000000  392.000000
mean      15.554220   76.010256    1.576531
std        2.750548    3.668093    0.805518
min        8.000000   70.000000    1.000000
25%       13.800000   73.000000    1.000000
50%       15.500000   76.000000    1.000000
75%       17.050000   79.000000    2.000000
max       24.800000   82.000000    3.000000
```

- The mpg's range is 37 and the mean is 23.
- The cylinder's range is 4 and the mean is 5.5 cylinders.
- The displacement's range is 387 and the mean is 194.4
- The horsepower's range is 184 and the mean is 104.5
- The weight's range is 3527 and the mean is 2977.6
- The accelearation range is 16.8 and the mean is 15.6
- The year's range is 12 and the mean is 76
- The origin's range is 2 and the mean is 1.6

1.3 Data Types

[ ]:
```python
print("Old\n",autodf.dtypes)

#Change Columns to Categorical
autodf.cylinders = autodf.cylinders.astype('category')
autodf.cylinders = autodf.cylinders.cat.codes

autodf.origin = pd.Categorical(autodf.origin)

print("\nNew\n", autodf.dtypes)
```

```
Old
 mpg             float64
cylinders         int64
displacement    float64
horsepower        int64
weight            int64
acceleration    float64
year            float64
origin            int64
```

```
name               object
dtype: object

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

1.4 NA Values

[ ]:
```python
#Count NA values
print(autodf.isnull().sum())

#Drop NA rows
autodf = autodf.dropna()

print("\n", autodf.isnull().sum())
print("Dimensions: ", autodf.shape)
```

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

 mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
year            0
origin          0
name            0
dtype: int64
Dimensions:  (389, 9)
```

1.5 Modify Columns

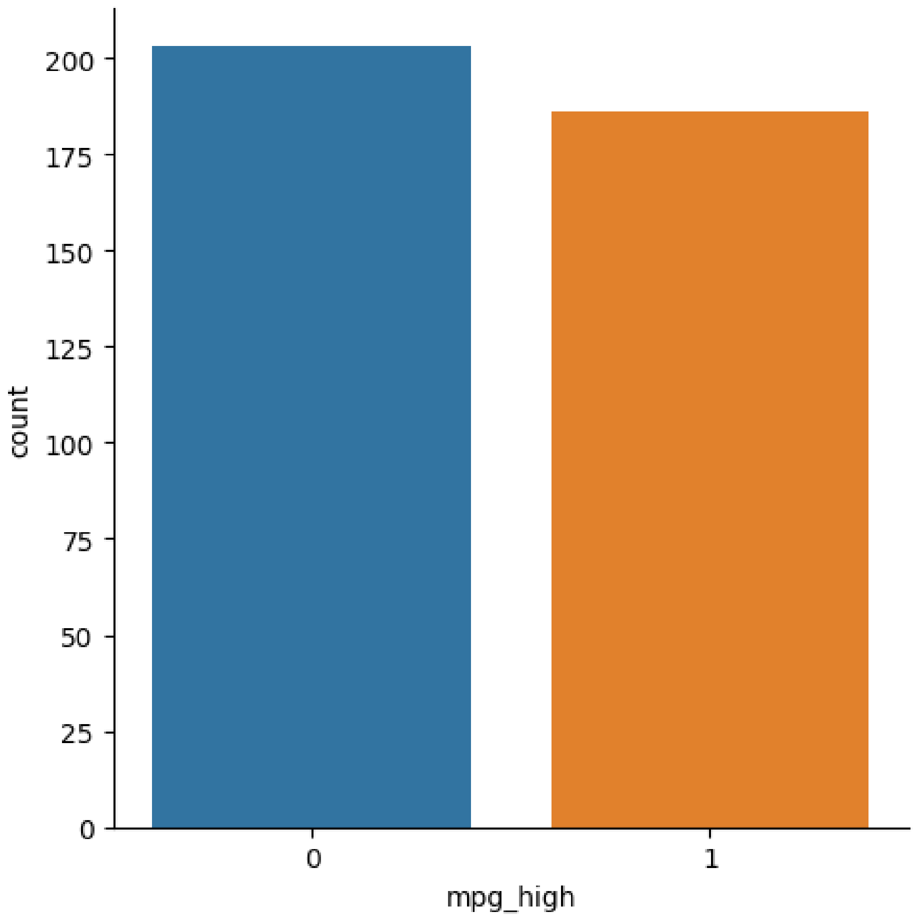
[ ]:
```python
#Create mpg high column
autodf['mpg_high'] = [1 if m > 23.445918 else 0 for m in autodf['mpg']]

#Drop mpg and name columns
autodf = autodf.drop(columns=['mpg', 'name'])
autodf.head()
```

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

   mpg_high
0         0
1         0
2         0
3         0
6         0
```

1.6 Graphical Data Exploration

[ ]:
```python
#MPG High Catplot
sb.catplot(x = "mpg_high", kind = "count", data=autodf)
```

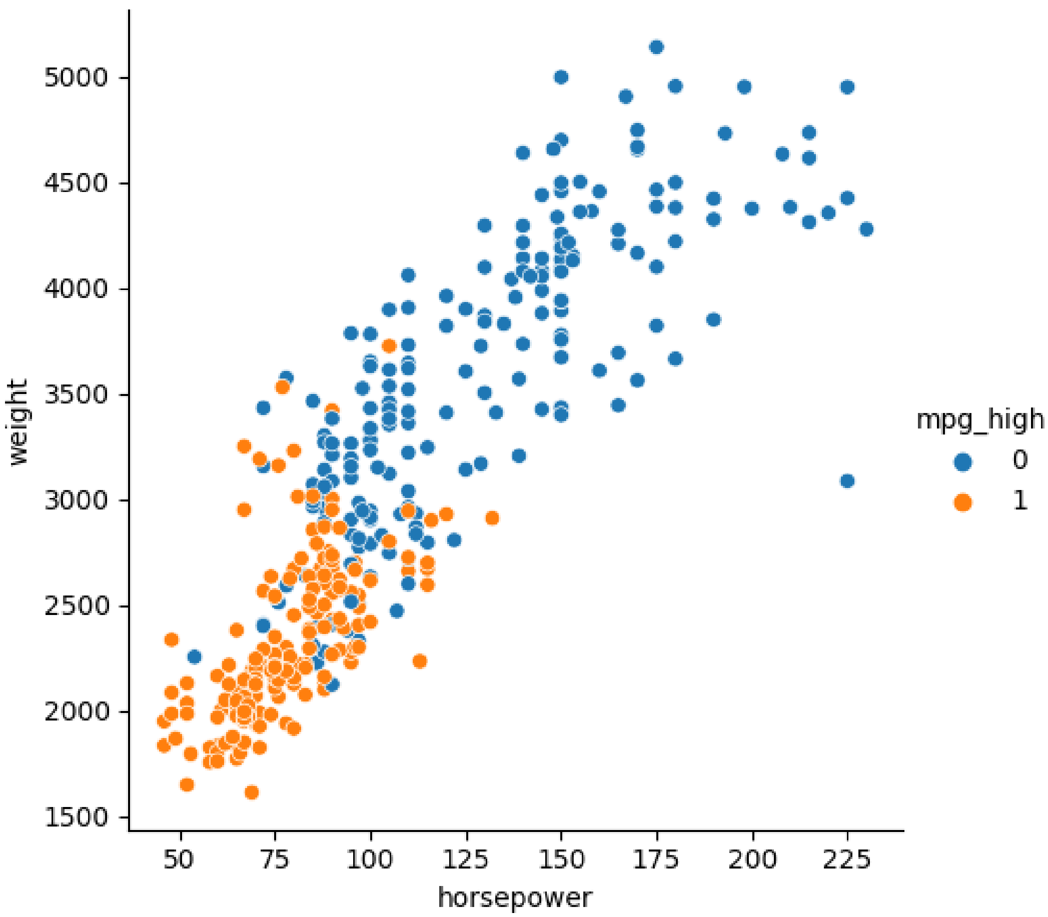
[ ]: <seaborn.axisgrid.FacetGrid at 0x7f6df2508df0>



It seems that mpg high seems evenly distributed, but there's more low_mpg cars

[ ]:
```
#Relplot
sb.relplot(x = "horsepower", y = "weight", hue = "mpg_high", data = autodf)
```

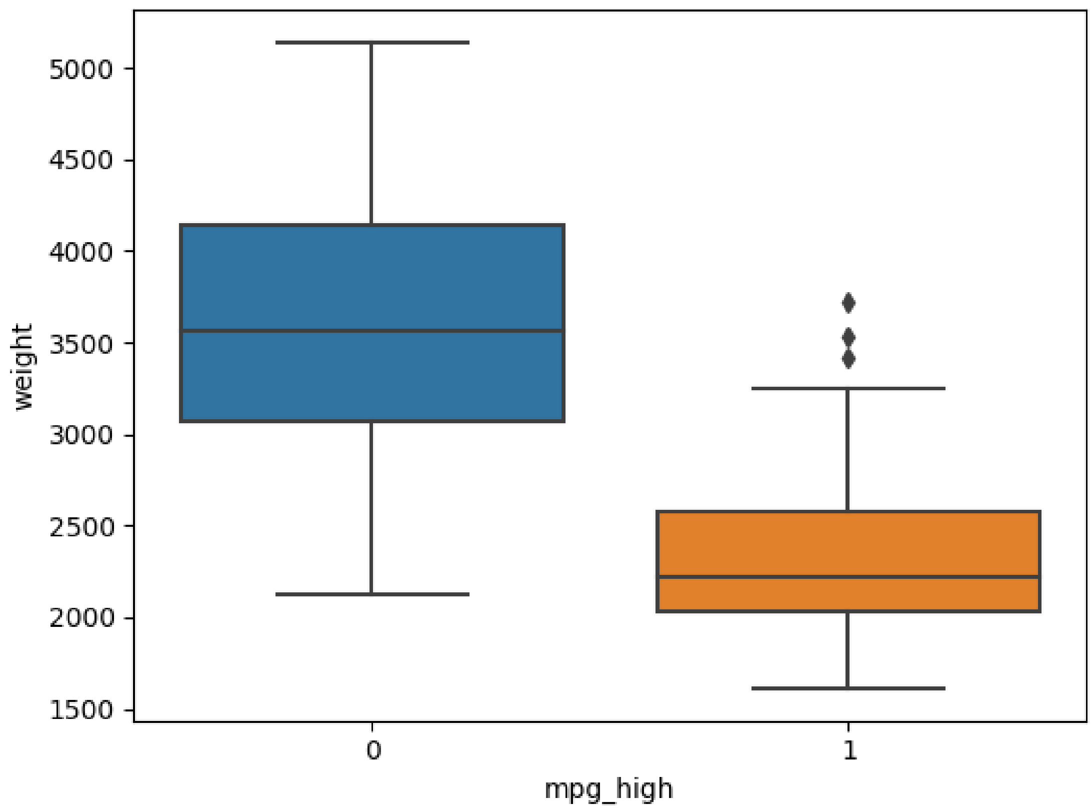
[ ]: <seaborn.axisgrid.FacetGrid at 0x7f6df265ce50>



The mpg_high seems clustered. High mpg is clustered on high horsepower and weight, while low mpg is clustered on low horsepower and weight. This might show that decision trees might be a good model to use.

[ ]:
```
#Boxplot
sb.boxplot(x = "mpg_high", y = "weight", data = autodf)
```

[ ]:
```
<AxesSubplot: xlabel='mpg_high', ylabel='weight'>
```



The low mpg cars seems to have a higher average weight than high mpg cars. The quartiles and range seems to be larger on the low mpg cars compared to the high mpg cars. Mpg high cars also seem to have a tiny bit of outliers. Most likely, the higher the weight of the car the more likely the car is mpg_low.

1.7 Split Data to Train and Test

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

#Obtain predictors and target columns
predictors = autodf.drop(columns=['mpg_high'])
target = autodf.mpg_high

#Split Data
predictorTrain, predictorTest, targetTrain, targetTest = ␣
 ↪train_test_split(predictors, target, test_size=.2, random_state=1234)

#Output Dimensions
print("Train Dimensions:", predictorTrain.shape)
print("Test Dimensions:", predictorTest.shape)
```

```
Train Dimensions: (311, 7)
Test Dimensions: (78, 7)
```

1.8 Logistic Regression

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

#Make model
lr = LogisticRegression()
lr.fit(predictorTrain, targetTrain)
print("Score: ", lr.score(predictorTrain, targetTrain))
```

```
Score:  0.9067524115755627

/home/pretaxend/.local/lib/python3.8/site-
packages/sklearn/linear_model/_logistic.py:444: ConvergenceWarning: lbfgs 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
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

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

#Predict
predictions = lr.predict(predictorTest)

#Metrics
print(classification_report(targetTest, predictions))
```

```
              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
```

1.9 Decision Tree

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

#Make Model
dt = DecisionTreeClassifier()
dt.fit(predictorTrain, targetTrain)
print("Score: ", dt.score(predictorTrain, targetTrain))
```

```
Score:  1.0
```

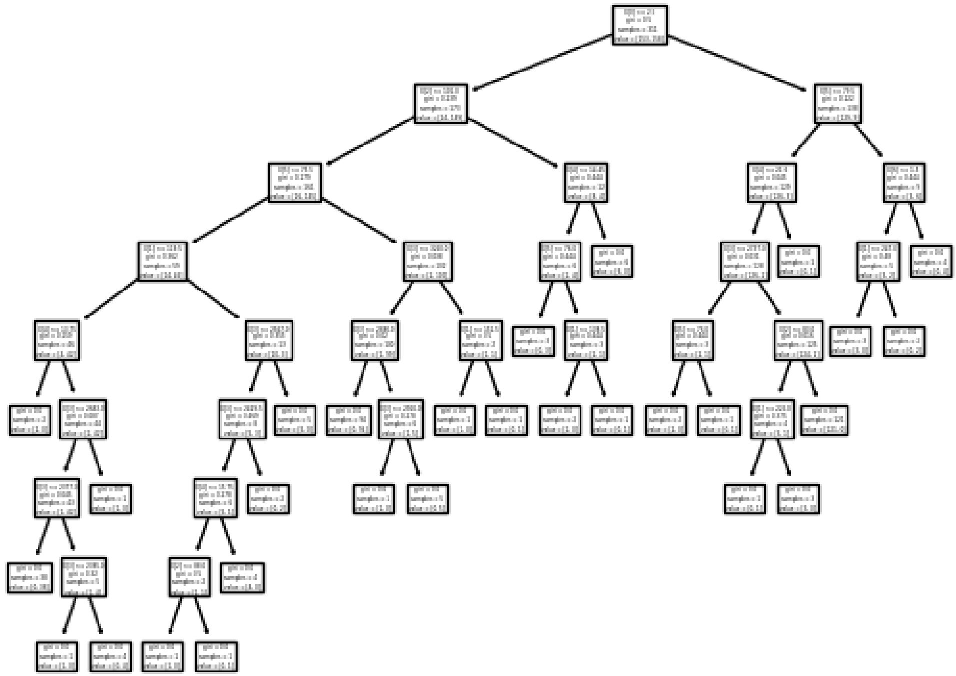
[ ]:
```
#Predict
predictions = dt.predict(predictorTest)

#Metrics
print(classification_report(targetTest, predictions))

#Plot Tree
tree.plot_tree(dt)
```

```
              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
```

[ ]:
```
[Text(0.6666666666666666, 0.9444444444444444, 'X[0] <= 2.5\ngini = 0.5\nsamples
= 311\nvalue = [153, 158]'),
 Text(0.4583333333333333, 0.8333333333333334, 'X[2] <= 101.0\ngini =
0.239\nsamples = 173\nvalue = [24, 149]'),
 Text(0.3055555555555556, 0.7222222222222222, 'X[5] <= 75.5\ngini =
0.179\nsamples = 161\nvalue = [16, 145]'),
 Text(0.16666666666666666, 0.6111111111111112, 'X[1] <= 119.5\ngini =
0.362\nsamples = 59\nvalue = [14, 45]'),
 Text(0.05555555555555555, 0.5, 'X[4] <= 13.75\ngini = 0.159\nsamples =
46\nvalue = [4, 42]'),
 Text(0.027777777777777776, 0.3888888888888889, 'gini = 0.0\nsamples = 2\nvalue
= [2, 0]'),
 Text(0.08333333333333333, 0.3888888888888889, 'X[3] <= 2683.0\ngini =
0.087\nsamples = 44\nvalue = [2, 42]'),
 Text(0.05555555555555555, 0.2777777777777778, 'X[3] <= 2377.0\ngini =
0.045\nsamples = 43\nvalue = [1, 42]'),
 Text(0.027777777777777776, 0.16666666666666666, 'gini = 0.0\nsamples =
```

```
38\nvalue = [0, 38]'),
 Text(0.08333333333333333, 0.16666666666666666, 'X[3] <= 2385.0\ngini =
0.32\nsamples = 5\nvalue = [1, 4]'),
 Text(0.05555555555555555, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nvalue
= [1, 0]'),
 Text(0.1111111111111111, 0.05555555555555555, 'gini = 0.0\nsamples = 4\nvalue =
[0, 4]'),
 Text(0.1111111111111111, 0.2777777777777778, 'gini = 0.0\nsamples = 1\nvalue =
[1, 0]'),
 Text(0.2777777777777778, 0.5, 'X[3] <= 2567.0\ngini = 0.355\nsamples =
13\nvalue = [10, 3]'),
 Text(0.25, 0.3888888888888889, 'X[3] <= 2429.5\ngini = 0.469\nsamples =
8\nvalue = [5, 3]'),
 Text(0.2222222222222222, 0.2777777777777778, 'X[4] <= 15.75\ngini =
0.278\nsamples = 6\nvalue = [5, 1]'),
 Text(0.19444444444444445, 0.16666666666666666, 'X[2] <= 88.0\ngini =
0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.16666666666666666, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nvalue
= [1, 0]'),
 Text(0.2222222222222222, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
 Text(0.25, 0.16666666666666666, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]'),
 Text(0.2777777777777778, 0.2777777777777778, 'gini = 0.0\nsamples = 2\nvalue =
[0, 2]'),
 Text(0.3055555555555556, 0.3888888888888889, 'gini = 0.0\nsamples = 5\nvalue =
[5, 0]'),
 Text(0.4444444444444444, 0.6111111111111112, 'X[3] <= 3250.0\ngini =
0.038\nsamples = 102\nvalue = [2, 100]'),
 Text(0.3888888888888889, 0.5, 'X[3] <= 2880.0\ngini = 0.02\nsamples =
100\nvalue = [1, 99]'),
 Text(0.3611111111111111, 0.3888888888888889, 'gini = 0.0\nsamples = 94\nvalue =
[0, 94]'),
 Text(0.4166666666666667, 0.3888888888888889, 'X[3] <= 2920.0\ngini =
0.278\nsamples = 6\nvalue = [1, 5]'),
 Text(0.3888888888888889, 0.2777777777777778, 'gini = 0.0\nsamples = 1\nvalue =
[1, 0]'),
 Text(0.4444444444444444, 0.2777777777777778, 'gini = 0.0\nsamples = 5\nvalue =
[0, 5]'),
 Text(0.5, 0.5, 'X[1] <= 151.5\ngini = 0.5\nsamples = 2\nvalue = [1, 1]'),
 Text(0.4722222222222222, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nvalue =
[1, 0]'),
 Text(0.5277777777777778, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
 Text(0.6111111111111112, 0.7222222222222222, 'X[4] <= 14.45\ngini =
0.444\nsamples = 12\nvalue = [8, 4]'),
 Text(0.5833333333333334, 0.6111111111111112, 'X[5] <= 76.0\ngini =
0.444\nsamples = 6\nvalue = [2, 4]'),
```

```
 Text(0.5555555555555556, 0.5, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(0.6111111111111112, 0.5, 'X[1] <= 138.5\ngini = 0.444\nsamples = 3\nvalue
= [2, 1]'),
 Text(0.5833333333333334, 0.3888888888888889, 'gini = 0.0\nsamples = 2\nvalue =
[2, 0]'),
 Text(0.6388888888888888, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
 Text(0.6388888888888888, 0.6111111111111112, 'gini = 0.0\nsamples = 6\nvalue =
[6, 0]'),
 Text(0.875, 0.8333333333333334, 'X[5] <= 79.5\ngini = 0.122\nsamples =
138\nvalue = [129, 9]'),
 Text(0.8055555555555556, 0.7222222222222222, 'X[4] <= 21.6\ngini =
0.045\nsamples = 129\nvalue = [126, 3]'),
 Text(0.7777777777777778, 0.6111111111111112, 'X[3] <= 2737.0\ngini =
0.031\nsamples = 128\nvalue = [126, 2]'),
 Text(0.7222222222222222, 0.5, 'X[5] <= 76.0\ngini = 0.444\nsamples = 3\nvalue =
[2, 1]'),
 Text(0.6944444444444444, 0.3888888888888889, 'gini = 0.0\nsamples = 2\nvalue =
[2, 0]'),
 Text(0.75, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.8333333333333334, 0.5, 'X[2] <= 83.0\ngini = 0.016\nsamples = 125\nvalue
= [124, 1]'),
 Text(0.8055555555555556, 0.3888888888888889, 'X[1] <= 225.0\ngini =
0.375\nsamples = 4\nvalue = [3, 1]'),
 Text(0.7777777777777778, 0.2777777777777778, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
 Text(0.8333333333333334, 0.2777777777777778, 'gini = 0.0\nsamples = 3\nvalue =
[3, 0]'),
 Text(0.8611111111111112, 0.3888888888888889, 'gini = 0.0\nsamples = 121\nvalue
= [121, 0]'),
 Text(0.8333333333333334, 0.6111111111111112, 'gini = 0.0\nsamples = 1\nvalue =
[0, 1]'),
 Text(0.9444444444444444, 0.7222222222222222, 'X[6] <= 1.5\ngini =
0.444\nsamples = 9\nvalue = [3, 6]'),
 Text(0.9166666666666666, 0.6111111111111112, 'X[1] <= 247.0\ngini =
0.48\nsamples = 5\nvalue = [3, 2]'),
 Text(0.8888888888888888, 0.5, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.9444444444444444, 0.5, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.9722222222222222, 0.6111111111111112, 'gini = 0.0\nsamples = 4\nvalue =
[0, 4]')]
```



1.10 Neural Network

```
[ ]: from sklearn import preprocessing
from sklearn.neural_network import MLPClassifier

#Normalize Data
scale = preprocessing.StandardScaler().fit(predictorTrain)
predictorTrainScaled = scale.transform(predictorTrain)
predictorTestScaled = scale.transform(predictorTest)

#Train
nnlbfgs = MLPClassifier(solver = 'lbfgs', hidden_layer_sizes = (3,2), max_iter␣
↪= 500, random_state = 1234)
nnlbfgs.fit(predictorTrainScaled, targetTrain)
print("Score: ", nnlbfgs.score(predictorTrainScaled, targetTrain))
```

```
Score:  0.9517684887459807
```

```
[ ]: #Predictions
predictions = nnlbfgs.predict(predictorTestScaled)

#Metrics
print(classification_report(targetTest, predictions))
```

```
              precision    recall  f1-score   support

           0       0.92      0.88      0.90        50
           1       0.80      0.86      0.83        28

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

```
[ ]: #Different Topology
nnsgd = MLPClassifier(solver = 'sgd', hidden_layer_sizes=(3,), max_iter=1000,␣
 ↪random_state=1234)
nnsgd.fit(predictorTrainScaled, targetTrain)

#Score
print("Score: ", nnsgd.score(predictorTrainScaled, targetTrain))
```

```
Score:  0.9003215434083601
```

```
[ ]: #Predictions
predictions = nnsgd.predict(predictorTestScaled)

#Metrics
print(classification_report(targetTest, predictions))
```

```
              precision    recall  f1-score   support

           0       0.93      0.80      0.86        50
           1       0.71      0.89      0.79        28

    accuracy                           0.83        78
   macro avg       0.82      0.85      0.83        78
weighted avg       0.85      0.83      0.84        78
```

The initial model has a higher accuracy most likely due to having more nodes. Most likely the model using the 'sgd' solver underfitted the data.

1.11 Analysis

The decision tree was the more accurate algorithm with an accuracy of 88%, most likely due to the different factors being clustered together. The highest precision for mpg_low is for logistic regression with .98, while the highest precision for mpg_high is for decision trees with .81. The best recall for mpg_low is decions trees with .88, while logistic regression has the best recall for mpg_high with .96.

The decision tree seems to be the better performing algorithm, most likely due to the 2 different factors being clustered together. The neural networks most likely underperformed due to not having as much data to work on. The logistic regression model seemed to perform well due to some predictors having linear relationships.

I personally prefer working in Python than R with ML. While R has better functionality regarding dataframes, the Pandas library still performs well and is easy to use. Sklearn seems to be much more user friendly for creating ML models than R.