```
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.neural_network import MLPClassifier
from sklearn import tree

# Input file name
input_file = "Auto.csv"

# Read data from csv into data variable
data = pd.read_csv(input_file, header = 0)
print('First few elements: ')
print(data.head())

# Print dimensions of data frame
print('Dimensions: ', data.shape)
```

```
First few elements:
    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)
```

```
# Describe columns
print(data[["mpg", "weight", "year"]].describe())
```

```
              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
```

Averages:

- Mpg: 23.445918
- Weight: 2977.584184
- Year: 76.010256

Range:

- Mpg: 37
- Weight: 3527
- Year: 12

```
# Explore the daata types
print('Original Datatypes:')
print(data.dtypes)

# Change the cylinders column to categorical using cat.codes
data['cylinders'] = data['cylinders'].astype('category').cat.codes

# Change the origin column to categorical without using cat.codes
data['origin'] = data['origin'].astype('category')

# Verify the changes with the dtypes attribute
print ('\nNew Datatypes:')
print (data.dtypes)
```

```
Original Datatypes:
mpg             float64
cylinders         int64
displacement    float64
horsepower        int64
weight            int64
acceleration    float64
year            float64
origin            int64
name             object
dtype: object

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

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


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

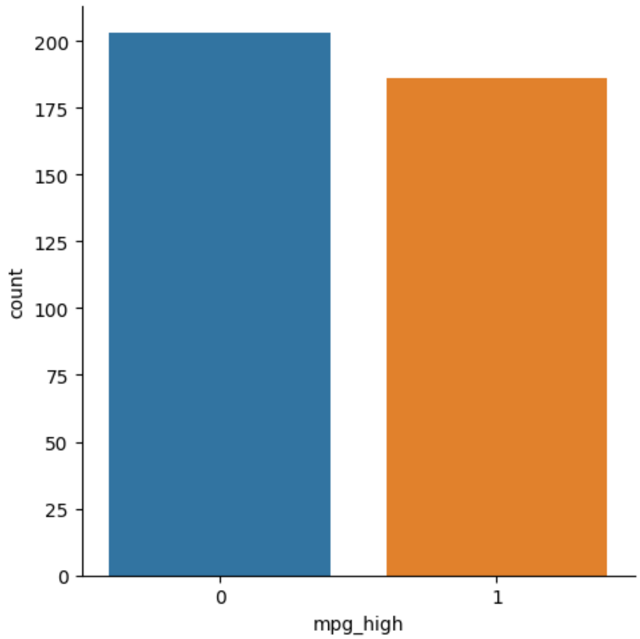
```
New dimensions: (389, 9)
```

```
# Make a new column (mpg_high) that is categorical
mpg_high = []
index = 0

for item in data.mpg:
  if data.mpg.mean() > item:
    mpg_high.insert(index, 0)
  else:
    mpg_high.insert(index, 1)
  index += 1

# Insert new column
data.loc[:, 'mpg_high'] = mpg_high

# Set to categorical
data.loc[:, 'mpg_high'] = data['mpg_high'].astype('category')

# Delete mpg and name columns
data = data.drop(columns=['mpg', 'name'])
print(data.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
<ipython-input-1170-89318f518b59>:13: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
  data.loc[:, 'mpg_high'] = mpg_high
<ipython-input-1170-89318f518b59>:16: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
  data.loc[:, 'mpg_high'] = data['mpg_high'].astype('category')
```

```
# Seaborn catplot on the mpg_high column
sb.catplot(data = data, x = 'mpg_high', kind = 'count')
```

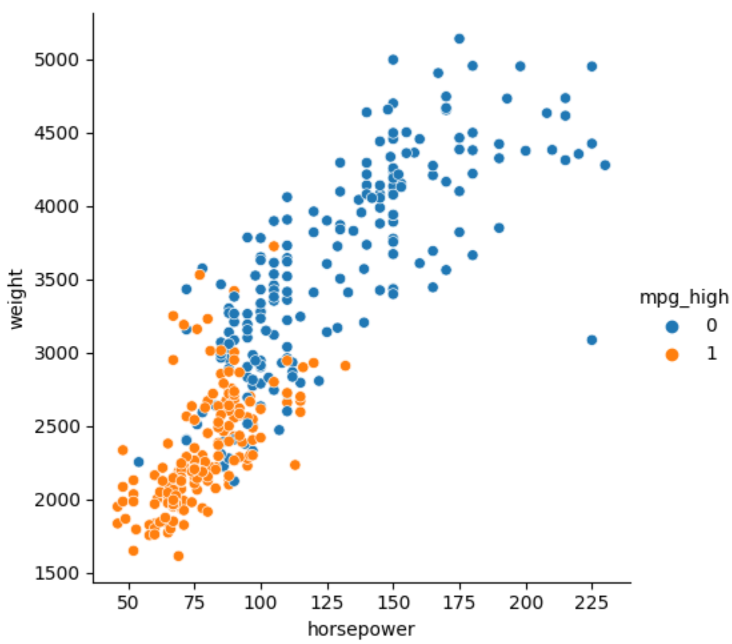
```
<seaborn.axisgrid.FacetGrid at 0x7f43935fb280>
```



From the catplot, we can see that there is nearly an equal amount of high and low mpg vehicles.

```
# Seaborn relplot with horsepower on the x axis, weight on the y axis, and setting hue to mpg_high
sb.relplot(data = data, x ='horsepower', y = 'weight', hue = 'mpg_high')
```

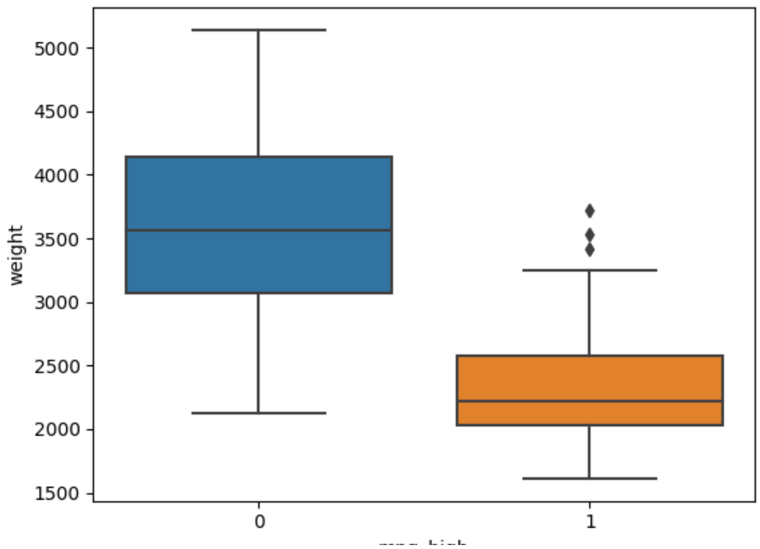
```
<seaborn.axisgrid.FacetGrid at 0x7f43931a0b20>
```



From the dotplot, we can see that heaver vehicles tend to have a higher horsepower, and vehicles that are heavy and have a high horsepower tend to have low mpg.

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

```
<Axes: xlabel='mpg_high', ylabel='weight'>
```



From the boxplot, we can see that the average weight of a high mpg vehicle is about 2250, and the average weight of a low mpg vehicle is about 3550.

```
# Split training and testing data
X = data.drop(columns = ['mpg_high'])
y = data['mpg_high']

print(X.head())

# Set random seed = 1234
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 1234, test_size = 0.20)

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

```
   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
(311, 7) (78, 7) (311,) (78,)
```

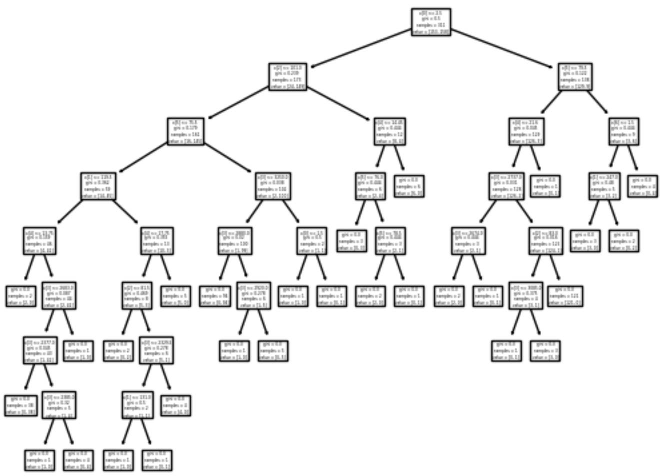
```
# Train a logistic regression model using solver lbfgs
clf = LogisticRegression(solver='lbfgs', max_iter= 175)
clf.fit(X_train, y_train)

# Print metrics using the classification report
pred = clf.predict(X_test)

print(classification_report(y_test, pred))
```

```
              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
```

```
# Train a decision tree
# Test and evaluate
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
pred = clf.predict(X_test)
print(classification_report(y_test, pred))
print(tree.plot_tree(clf))
```

```
              precision    recall  f1-score   support

           0       0.90      0.92      0.91        50
           1       0.85      0.82      0.84        28

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

```
[Text(0.6507352941176471, 0.9444444444444444, 'x[0] <= 2.5\ngini = 0.5\nsamples = 311\nvalue = [153, 158]'), Text(0.4338235294117647, 0
```



```
# train a neural network, choosing a network topology of your choice
# test and evaluate
classifier = MLPClassifier(hidden_layer_sizes = (6, 5, 4, 3, 2), random_state = 1234, max_iter= 1000)
classifier.fit(X_train, y_train)
pred = classifier.predict(X_test)

print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.93      0.82      0.87        50
           1       0.74      0.89      0.81        28

    accuracy                           0.85        78
   macro avg       0.83      0.86      0.84        78
weighted avg       0.86      0.85      0.85        78
```

```
classifier2 = MLPClassifier(hidden_layer_sizes = (5, 3), random_state = 1234, max_iter= 1000)
classifier2.fit(X_train, y_train)
pred2 = classifier2.predict(X_test)

print(classification_report(y_test, pred2))
```

```
              precision    recall  f1-score   support

           0       0.95      0.84      0.89        50
           1       0.76      0.93      0.84        28

    accuracy                           0.87        78
   macro avg       0.86      0.88      0.87        78
weighted avg       0.89      0.87      0.87        78
```

The topology of the first neural network was designed to increase the likelihood of overfitting by setting up a very complex network architecture compared to the second neural network, which was much simplier by comparison. The first neural network (overfit) slightly underperformed the second in all metrics because it was overfit.

Analysis

a. The decision tree performed the best by far, followed by the neural network and finally the logistic regression.
b. The decision tree outperformed in all metrics. The neural network outperformed the logistic regression in all metrics.
c. I believe the decision tree outperformed the other metrics because there weren't too few features for the logistic regression to shine, but there also weren't so many features for the neural network to shine. The feature count was best-suited for the decision tree because of this.

d. I personally much prefer working with python when it comes to implementing machine learning algorithms. The format of the language feels much more familiar to other languages I have worked with (C++, Java) and each of the libraries feels deliberate. R has its perks, such as having so many built in functionalities available, but it feels less intuitive to me because of this. One aspect of R that I appreciate more, however, are the many built-in options for displaying data. Seaborn is nice, but I do think R was less confusing in that aspect.