First lets read in the data

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sb
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.tree import DecisionTreeClassifier
from sklearn import preprocessing
from sklearn.metrics import classification_report
df = pd.read_csv("Auto.csv")
print(df.head())
print('\nDimensions of data frame:', df.shape)
```

```
    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 of data frame: (392, 9)
```

Data exploration

In [2]:
```
df[['mpg', 'weight', 'year']].describe()
```

Out[2]:

	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

The range of the columns in order are as follows (37.6, 3527, 10). The average of each of the columns in order are as follows (23.445918, 2977.584184, 76.010256).

Now we're going to do some more data exploration

In [3]:
```
df.dtypes
```

Out[3]:
```
mpg             float64
cylinders         int64
displacement    float64
horsepower        int64
weight            int64
acceleration    float64
year            float64
origin            int64
name             object
dtype: object
```

Lets change some of these types and output them again

In [4]:
```
df.cylinders = df.cylinders.astype('category').cat.codes
df['origin'] = df['origin'].astype('category')
df.dtypes
```

Out[4]:
```
mpg              float64
cylinders           int8
displacement     float64
horsepower         int64
weight             int64
acceleration     float64
year             float64
origin          category
name              object
dtype: object
```

I tried using df.cylinders = df.cylinders.astype('category').cat.codes to change the first one, but for some reason it would not change. I resigned to just using the same method for both as to continue with the assignment.

Now lets get rid of the nows with NA

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    1
year            2
origin          0
name            0
dtype: int64
```

In [6]:
```
df = df.dropna()
```

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
year            0
origin          0
name            0
dtype: int64
```

In [8]:
```
print('\nDimensions of data frame:', df.shape)
```

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

Lets add the mpg_high column

In [9]:
```
df['mpg_high'] = np.where(df['mpg'] > df['mpg'].mean(), 1, 0)
df['mpg_high'] = df['mpg_high'].astype('category')
```

Now lets remove the mpg column as well as the name column

In [10]:
```
df = df.drop(columns = ['mpg', 'name'])
df.head()
```

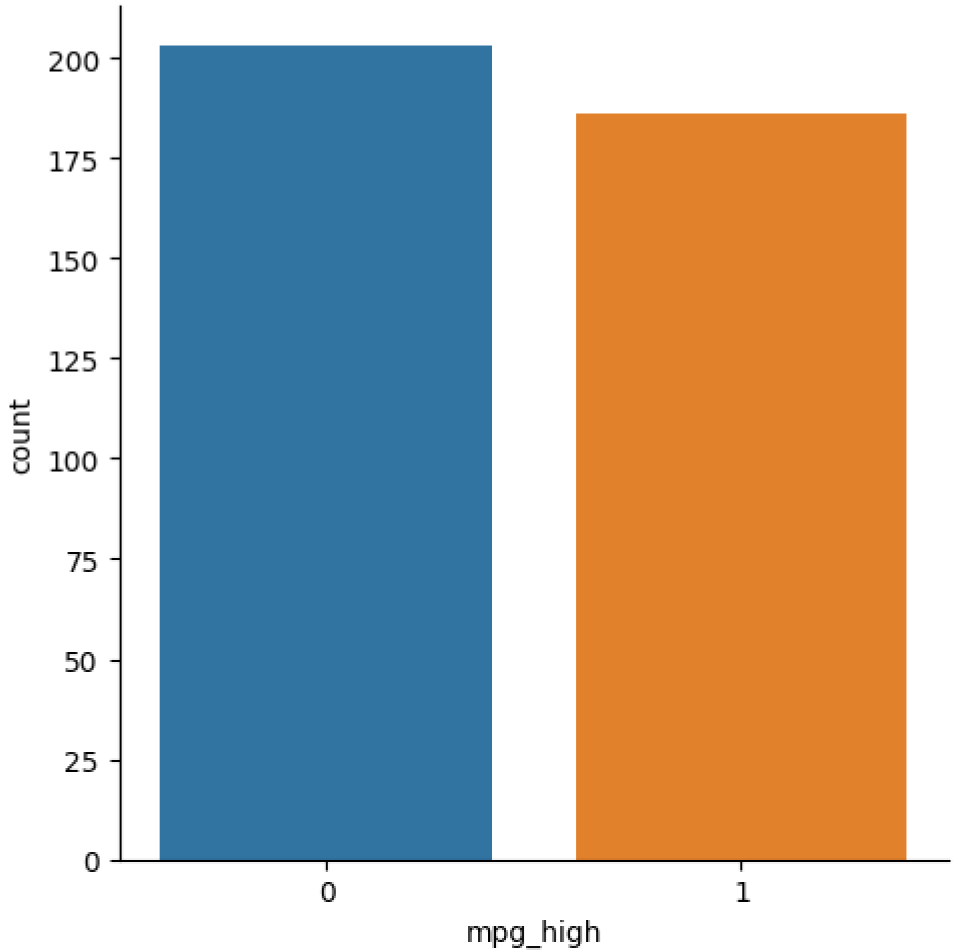
Out[10]:

	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

Data Exploration with graphs

In [11]:
```
sb.catplot(x = "mpg_high", kind = "count", data = df)
```

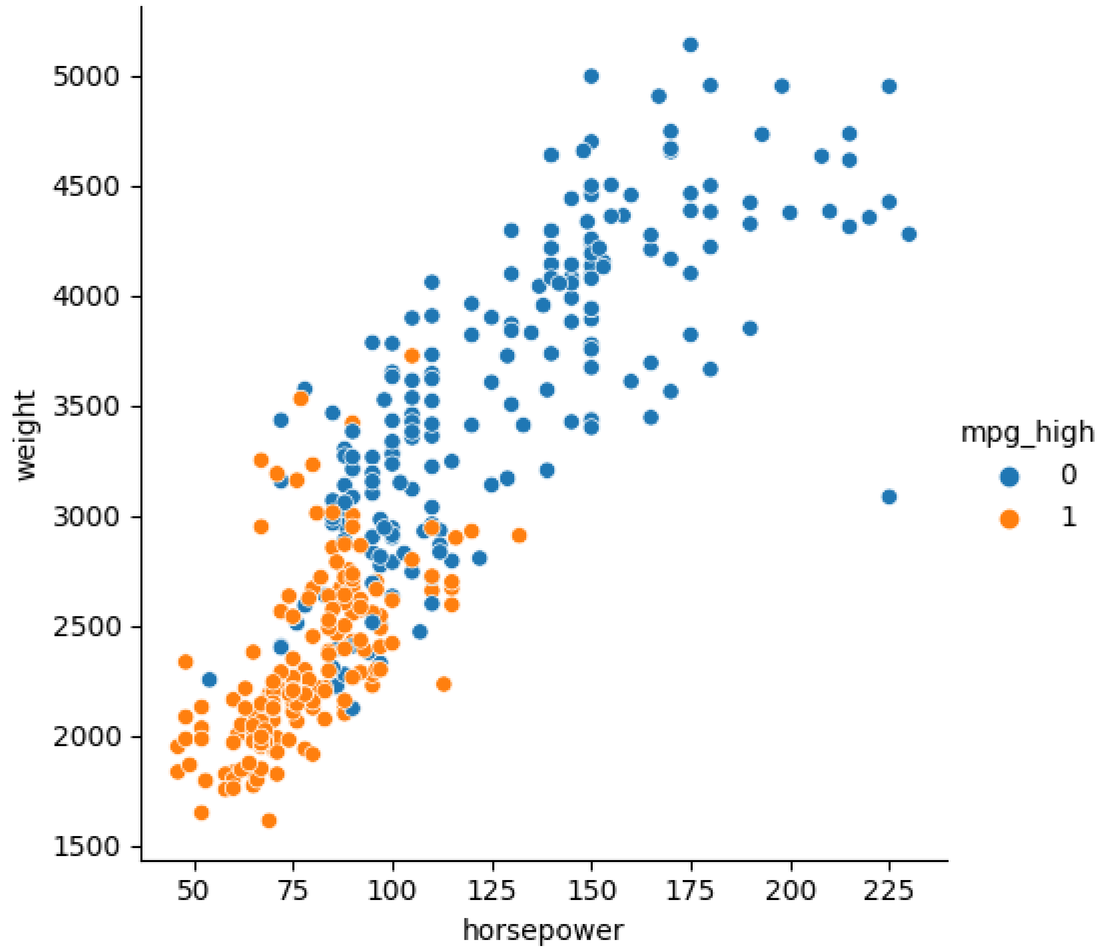
Out[11]: <seaborn.axisgrid.FacetGrid at 0x1f2bfa363b0>



Using this graph we can see that there are slightly more cars that have a less than average mpg, which means that the cars with higher than average mpg near the top must be a bit higher than the cars with lower than average mpg near the bottom.

In [12]:
```
sb.relplot(x = "horsepower", y = "weight", data = df, hue = "mpg_high")
```

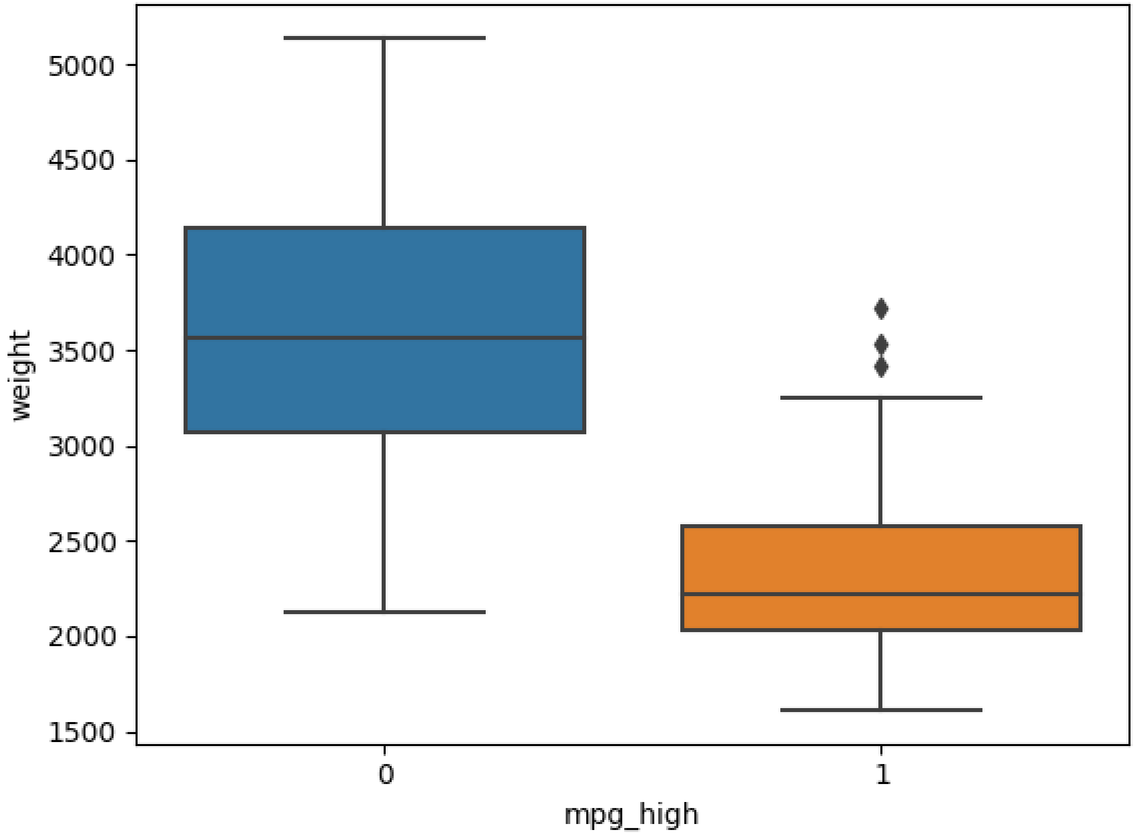
Out[12]: <seaborn.axisgrid.FacetGrid at 0x1f2bfa85ed0>



This graph tells us that heavier vehicles tend to have higher horsepower and also consume more fuel.

In [13]:
```
sb.boxplot(x = "mpg_high", y = "weight", data = df)
```

Out[13]: <AxesSubplot: xlabel='mpg_high', ylabel='weight'>



This graph tells us that of the vehicles that use more gas, they are heavier and the lower boundary for these vehicles weight is around the average weight of a lighter vehicle. Also the vehicles that weigh more have a higher range of weights compared to those that weigh less and consume less gas. There are also a couple of outliers in the vehicles that use less gas that are heavier than the others. These are probably SUV's where as the heavier vehicles are probably trucks.

Lets divide the data into test and train

In [14]:
```
X = df.loc[:, df.columns != 'mpg_high']
y = df.mpg_high
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2, random_state =
print('\nDimensions of train data frame:', X_train.shape)
print('\nDimensions of test frame:', X_test.shape)
```

```
Dimensions of train data frame: (311, 7)

Dimensions of test frame: (78, 7)
```

Now lets train a Logistic Regression model

In [15]:
```
clf = LogisticRegression(max_iter = 400, random_state = 1234)
clf.fit(X_train, y_train)
clf.score(X_train, y_train)
pred = clf.predict(X_test)
```

In [16]:
```
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))
```

```
accuracy score:  0.8974358974358975
precision score:  0.7777777777777778
recall score:  1.0
f1 score:  0.8750000000000001
```

It seems that using Logistic Regression we can get a pretty good idea of if a car will consume a lot of gas or not based on the other factors provided such as weight, horsepower, year, etc.

Now lets try a Decision Tree

In [17]:
```
clf = DecisionTreeClassifier(random_state = 1234)
clf.fit(X_train, y_train)
pred = clf.predict(X_test)
```

In [18]:
```
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))
```

```
accuracy score:  0.9230769230769231
precision score:  0.8666666666666667
recall score:  0.9285714285714286
f1 score:  0.896551724137931
```

here the decision tree got a better accuracy and a much better precision score but about 8 percent less on the recall score. The precision score makes sense to me since DT puts the data points into sort of "buckets" so they would natrually be close together comapred to Logistic Regression in some cases.

Now lets do a Neural Network

First lets normalize the data.

In [19]:
```
scaler = preprocessing.StandardScaler().fit(X_train)

X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Now lets train the NN

In [20]:
```
clf = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5, 2), max_iter=1000, random_s
clf.fit(X_train_scaled, y_train)
pred = clf.predict(X_test_scaled)
```

Lets see how the NN did

In [21]:
```
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.93      0.86      0.90        50
           1       0.78      0.89      0.83        28

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

The NN here actually did slightly worse than Logistic Regression, let's try with some different settings and see how this changes.

In [22]:
```
clf = MLPClassifier(solver='adam', hidden_layer_sizes=(100,), max_iter=2000, random_st
clf.fit(X_train_scaled, y_train)
pred = clf.predict(X_test_scaled)
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.93      0.86      0.90        50
           1       0.78      0.89      0.83        28

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

It seems that changing some of the settings had little to no difference on the outcome.

In [23]:
```
clf = MLPClassifier(solver='sgd', hidden_layer_sizes=(100,), max_iter=1000, random_sta
clf.fit(X_train_scaled, y_train)
pred = clf.predict(X_test_scaled)
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
```

Analysis

Out of the models that we attempted here the best results by about 4 percent was the Decision Tree. I partially expected better outcomes from the NN, perhaps they would perform better with some other combination of settings. The DT doing better than LR is no surprise, as it seemed from the graphs that the data was grouped quite well on one axis but maybe not the others. The recall score of LR makes me think that it may have overfitted a bit or perhaps the outliers negatively affected it more. I personally really liked using sklearn as opposed to R. I think that here you can sort of feel that it was written for Computer Scientists where R feels that it was made for Data Scientists or Statisticians. Especially when it comes to manipulating the data, I thought that writing the python code felt much more natural.