Read in Auto.csv

In [1]:
```
import pandas as pd
df = pd.read_csv('Auto.csv')
print(df.head())
print('\nShape of Auto.csv: {}'.format(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

Shape of Auto.csv: (392, 9)
```

Print out data on the mpg, weight, and year columns

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

mpgRange = dfSum['mpg']['max'] - dfSum['mpg']['min']
weightRange = dfSum['weight']['max'] - dfSum['weight']['min']
yearRange = dfSum['year']['max'] - dfSum['year']['min']

print('\nMPG Range: {}\nMPG Average: {}'.format(mpgRange, dfSum['mpg']['mean']))
print('\nWeight Range: {}\nWeight Average: {}'.format(weightRange, dfSum['weight'][
print('\nYear Range: {}\nYear Average: {}'.format(yearRange, dfSum['year']['mean'])
```

```
              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

MPG Range: 37.6
MPG Average: 23.445918367346938

Weight Range: 3527.0
Weight Average: 2977.5841836734694

Year Range: 12.0
Year Average: 76.01025641025642
```

Explore Column Data Types

Original column data types

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
```

In [4]:
```
df.cylinders = df.cylinders.astype('category').cat.codes
print('Cylinders type: {}'.format(df['cylinders'].dtype))
```

```
Cylinders type: int8
```

In [5]:
```
df.origin = df.origin.astype('category')
print('Origin type: {}'.format(df['origin'].dtype))
```

```
Origin type: category
```

Modified column data types

In [6]:
```
df.dtypes
```

Out[6]:
```
mpg              float64
cylinders           int8
displacement     float64
horsepower         int64
weight             int64
acceleration     float64
year             float64
origin          category
name              object
dtype: object
```

Handle NAs in the Data

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

Out[7]:
```
mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    1
year            2
origin          0
name            0
dtype: int64
```

In [8]:
```
print('Original dimensions: {}'.format(df.shape))
df = df.dropna()
print('Dimensions after removing NAs: {}'.format(df.shape))
```

```
Original dimensions: (392, 9)
Dimensions after removing NAs: (389, 9)
```

Modify Columns

Add new mpg_high column, which indicates the vehicle's mpg is higher than the average mpg

In [9]:
```
mpgSum = df.loc[:, 'mpg'].describe()
mpgAverage = mpgSum['mean']

df = df.assign(mpg_high=lambda x: x.mpg > mpgAverage)
df.mpg_high = df.mpg_high.astype('category').cat.codes
```

Remove mpg and name columns

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

Out[10]:

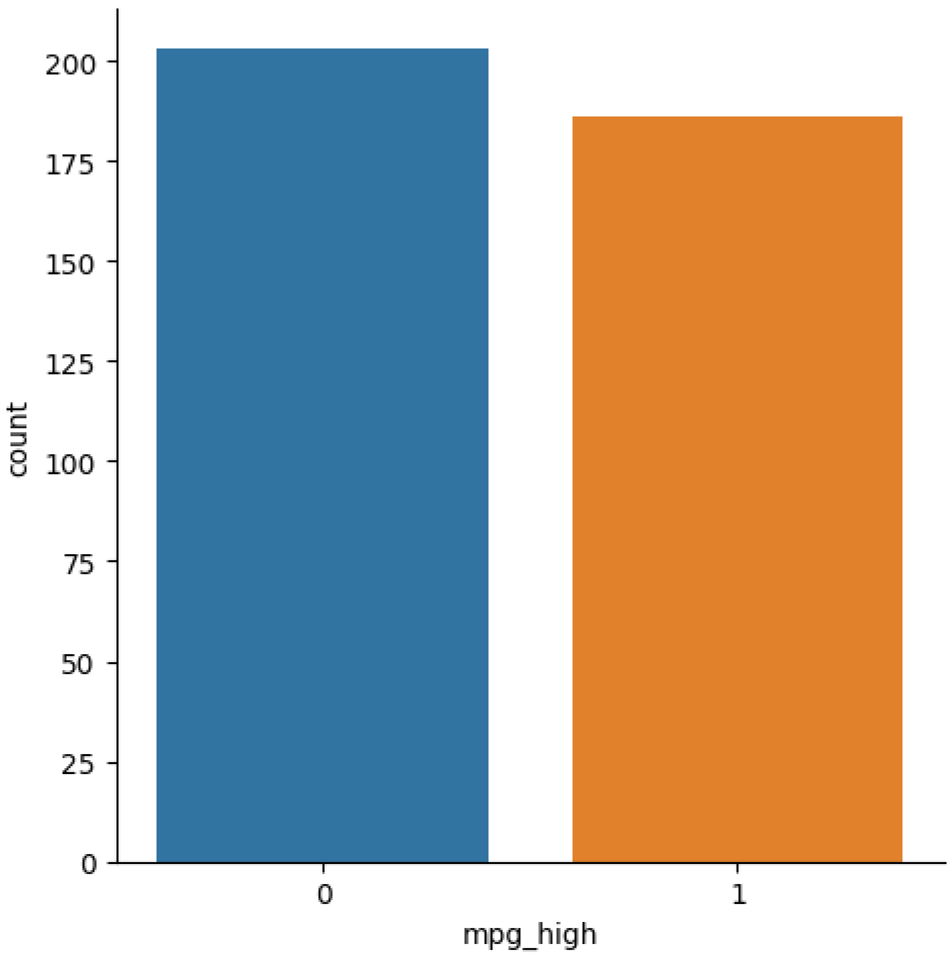
	cylinders	displacement	horsepower	weight	acceleration	year	origin	mpg_high
0	4	307.0	130	3504	12.0	70.0	1	0
1	4	350.0	165	3693	11.5	70.0	1	0
2	4	318.0	150	3436	11.0	70.0	1	0
3	4	304.0	150	3433	12.0	70.0	1	0
6	4	454.0	220	4354	9.0	70.0	1	0

Data Exploration with Graphs

Catplot

In [11]:
```
import seaborn as sb
sb.catplot(x='mpg_high', kind='count', data=df)
```

Out[11]:
```
<seaborn.axisgrid.FacetGrid at 0x173c023b250>
```

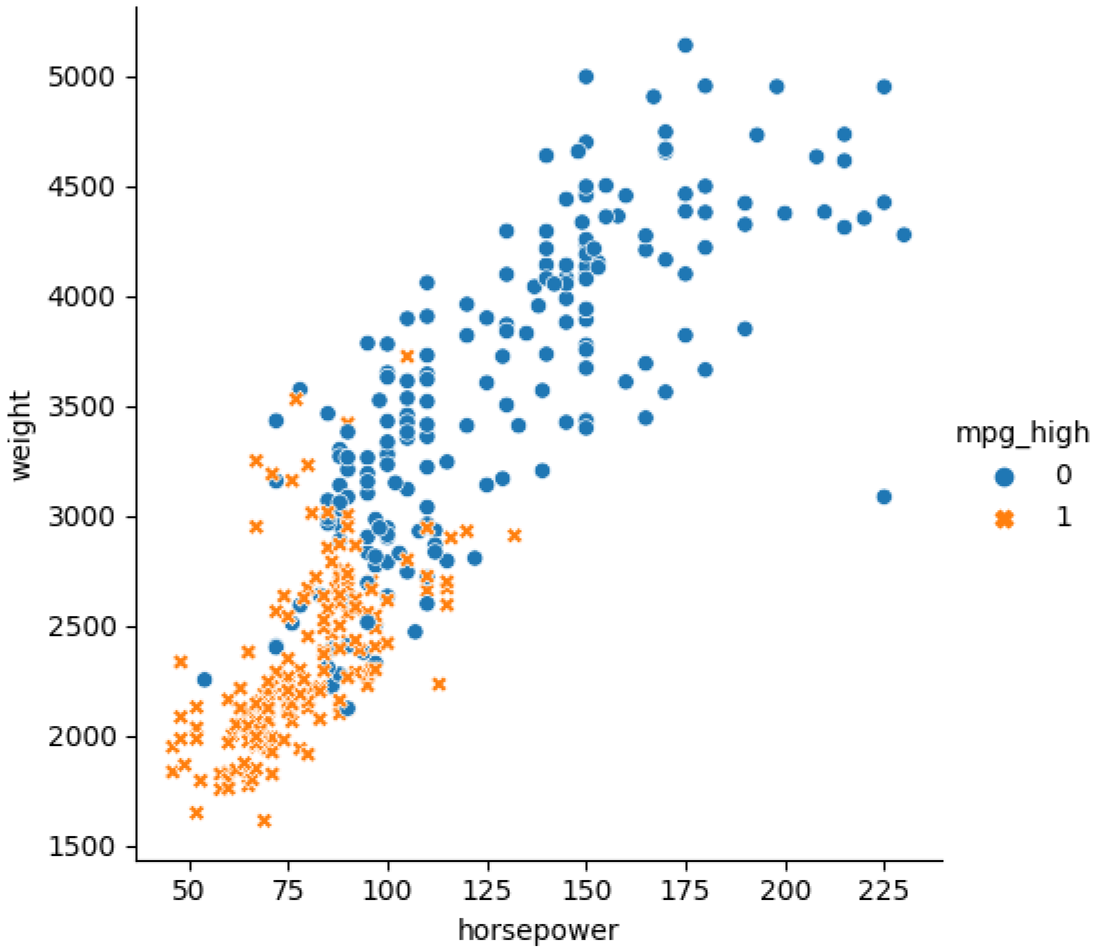


From the catplot above, it can be seen that there are roughly equal amounts of vehicles with high/low mpg. This makes sense, since mpg_high was assigned based on a vehicle's mpg compared to the average mpg from the dataset

Relplot

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

Out[12]:
```
<seaborn.axisgrid.FacetGrid at 0x173c02ae750>
```

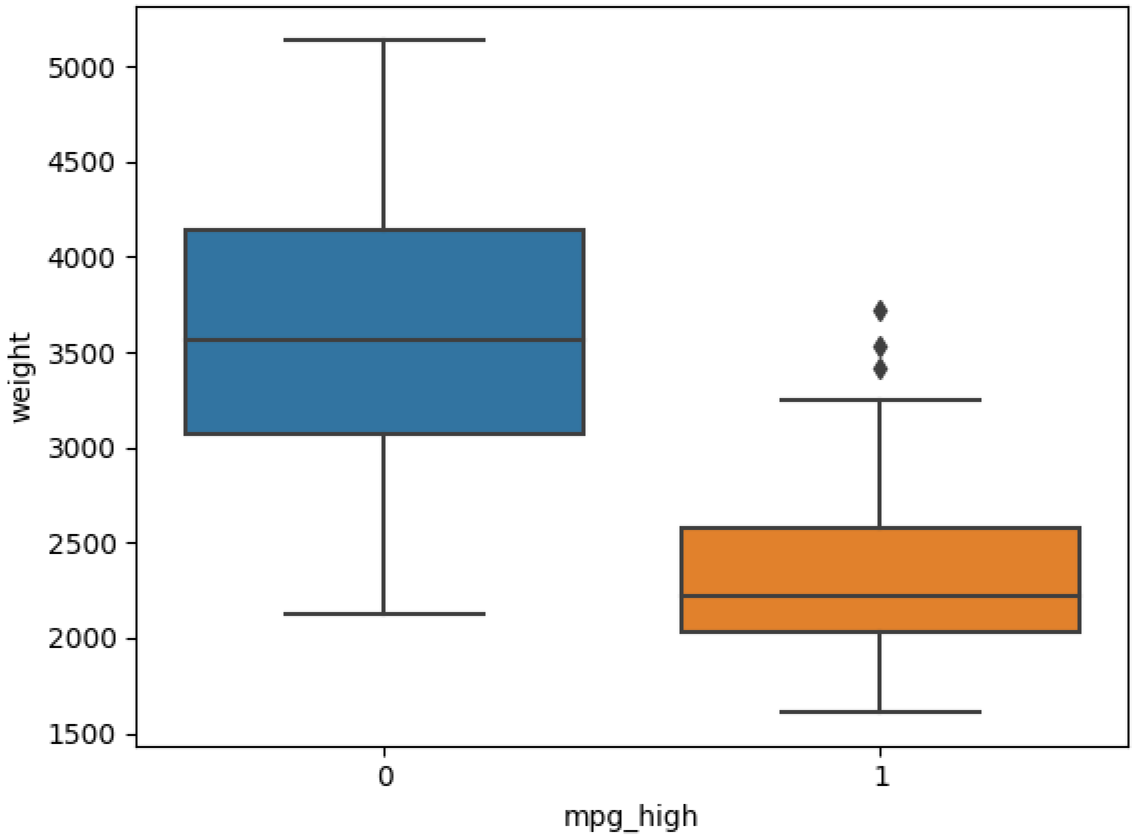


From the relplot above, it can be seen that good mpg is based on a vehicle's horsepower and weight. Roughly speaking, it seems that most vehicles with a weight under 3000 and horsepower under 125 were classified as mpg_high. Although there are outliers on both sides of this classification, it largely holds true

Boxplot

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

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



From the boxplot above, it can be seen that the average weight for mpg_high vehicles is nearly half of non-mpg_high vehicles. Also, it is interesting how wide the range of non-mpg_high vehicles' weight is; this leads me to believe that, although it plays a role in determining a vehicle's mpg, perhaps a vehicle's weight isn't as influential as other factors.

Train/Test Split

Random sample 80/20 split, drop mpg_high column from both sets

In [14]:
```
from sklearn.model_selection import train_test_split
X = df.loc[:, ['cylinders', 'displacement', 'horsepower', 'weight', 'acceleration',
y = df.mpg_high

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
print('Dimensions of X_train: {}'.format(X_train.shape))
print('Dimensions of X_test: {}'.format(X_test.shape))
```

```
Dimensions of X_train: (311, 7)
Dimensions of X_test: (78, 7)
```

Logistic Regression

In [18]:
```
X_train.cylinders = X_train.cylinders.astype('category').cat.codes
X_train.displacement = X_train.displacement.astype('category').cat.codes
X_train.horsepower = X_train.horsepower.astype('category').cat.codes
```

```
X_train.weight = X_train.weight.astype('category').cat.codes
X_train.acceleration = X_train.acceleration.astype('category').cat.codes
X_train.year = X_train.year.astype('category').cat.codes
X_train.origin = X_train.origin.astype('category').cat.codes

X_test.cylinders = X_test.cylinders.astype('category').cat.codes
X_test.displacement = X_test.displacement.astype('category').cat.codes
X_test.horsepower = X_test.horsepower.astype('category').cat.codes
X_test.weight = X_test.weight.astype('category').cat.codes
X_test.acceleration = X_test.acceleration.astype('category').cat.codes
X_test.year = X_test.year.astype('category').cat.codes
X_test.origin = X_test.origin.astype('category').cat.codes
```

In [19]:
```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression()
clf.fit(X_train, y_train)
clf.score(X_train, y_train)
```

Out[19]: 0.9131832797427653

In [20]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

pred = clf.predict(X_test)

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:  0.3974358974358974
Precision score:  0.37333333333333335
Recall score:  1.0
f1 score:  0.5436893203883496
```

Decision Tree

In [22]:
```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

pred = clf.predict(X_test)
```

In [23]:
```
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:  0.48717948717948717
precision score:  0.40625
recall score:  0.9285714285714286
f1 score:  0.5652173913043478
```

Neural Network

In [33]:
```
from sklearn import preprocessing

scaler = preprocessing.StandardScaler().fit(X_train)

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

Guidelines on the number of hidden nodes

(1) 1 - 7 (2) 3 (3) <14

For my first topology, I decided to try a (3,1) layout

In [89]:
```
from sklearn.neural_network import MLPClassifier

clf = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(3,1), max_iter=500, random_
clf.fit(X_train_scaled, y_train)

pred = clf.predict(X_test_scaled)
```

In [90]:
```
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_test, pred))
confusion_matrix(y_test, pred)
```

```
              precision    recall  f1-score   support

           0       1.00      0.22      0.36        50
           1       0.42      1.00      0.59        28

    accuracy                           0.50        78
   macro avg       0.71      0.61      0.48        78
weighted avg       0.79      0.50      0.44        78
```

Out[90]:
```
array([[11, 39],
       [ 0, 28]], dtype=int64)
```

Next, I try more hidden nodes with a (5,2) layout

In [91]:
```
clf = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5,2), max_iter=1500, random
clf.fit(X_train_scaled, y_train)

pred = clf.predict(X_test_scaled)
```

In [92]:
```
print(classification_report(y_test, pred))
confusion_matrix(y_test, pred)
```

```
              precision    recall  f1-score   support

           0       1.00      0.08      0.15        50
           1       0.38      1.00      0.55        28

    accuracy                           0.41        78
   macro avg       0.69      0.54      0.35        78
weighted avg       0.78      0.41      0.29        78
```

Out[92]:
```
array([[ 4, 46],
       [ 0, 28]], dtype=int64)
```

The first topology was a better model, having a higher average. This is likely due to the fact that the training was done on a small dataset, so the less complex topology would be a better fit. The second topology likely picked up on too much noise in the data.

Analysis

Of all the algorithms tested, it seems the first Neural Network performed the best, having an accuracy of 0.50. I am a little surprised that even the best performing algorithm only had an accuracy of 0.50.

Logistic Regression

Accuracy score: 0.3974358974358974

Precision score: 0.37333333333333335

Recall score: 1.0

Decision Tree

Accuracy score: 0.48717948717948717

Precision score: 0.40625

Recall score: 0.9285714285714286

Neural Network

Topology 1

```
    precision    recall  f1-score   support

         0       1.00      0.22      0.36        50
         1       0.42      1.00      0.59        28

  accuracy                           0.50        78
```

Topology 2

```
precision    recall  f1-score   support

       0       1.00      0.08      0.15        50
       1       0.38      1.00      0.55        28

accuracy                           0.41        78
```

Since the boundary between mpg_high and non-mpg_high was a little muddy, as could be seen in the relplot, perhaps the overtraining on the neural network worked well for random assignment.

Personally, I enjoyed working with python and sklearn much more than R. Although it was somewhat simpler in R since R was designed for machine learning, I absolutely despise the syntax in R. In my opinion, the python/sklearn approach was much more readable and enjoyable.