Homework 8

Author: Anthony Martinez | amm180005 \ Date: 11/19/21

1. Read the Auto data (5 points)

```
a. use pandas to read the data (available on Piazza)
b. print the first few rows
c. print the dimensions of the data
```

In [172]:
```
import pandas as pd

# a. use pandas to read the data
df = pd.read_csv('Auto.csv')

# b. print the first few rows
df.head()

# c. print the dimensions of the data
print('Dimensions of the data frame: ', df.shape)
```

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

2. Some data exploration with code (10 points)

```
a. use describe() on the mpg, weight, and year columns
b. write comments indicating the range and average of each column

Commentary:
mpg: range = (9.0, 46.0) | average = 23.445918
weight: range = (1613.0, 5140.0) | average = 2977.584184
year: range = (70.0, 82.0) | average = 76.010256
```

In [173]:
```
# a. use describe() on the mpg, weight, and year columns
df.loc[:, ['mpg', 'weight', 'year']].describe()
```

Out[173]:

	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

3. Explore data types (10 points)

```
a. check the data types of all columns
b. change the cylinders column to categorical (use cat.codes)
c. change the origin column to categorical (don’t use cat.codes)
d. verify the changes with the dtypes attribute
```

In [174]:
```
# a. check the data types of all columns
print('Before column conversions:\n')
print(df.dtypes)

# b. change the cylinders column to categorical (use cat.codes)
df.cylinders = df.cylinders.astype('category').cat.codes
# using cat.codes will mean data type will show up as int8 rather than categor
y

# c. change the origin column to categorical (don't use cat.codes)
df.origin = df.origin.astype('category')

# d. verify the changes with the dtypes attribute
print('\nAfter column conversions:\n')
print(df.dtypes)
```

```
Before column conversions:

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

After column conversions:

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

4. Deal with NAs (5 points)

a. delete rows with NAs
b. print the new dimensions

In [175]:
```
# check for NAs
df.isnull().sum()

print('Acceleration and year are the only columns with NAs')

# a. delete rows with NAs
df = df.dropna()

# b. print the new dimensions | new dimensions indicate that the rows were del
eted
print('\nDimensions of data frame:', df.shape)
```

```
Acceleration and year are the only columns with NAs

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

5. Modify columns (10 points)

a. make a new column, mpg_high, which is categorical:
   i. the column == 1 if mpg > average mpg, else == 0
b. delete the mpg and name columns
c. print the first few rows of the modified data frame

In [176]:
```
import numpy as np

# a. make a new column, mpg_high, which is categorical: the column == 1 if mpg
> average mpg, else == 0
average_mpg = np.mean(df.mpg)
df['mpg_high'] = np.where(df['mpg'] > average_mpg , 1, 0)

# b. delete the mpg and name columns
df1 = df.copy()
df1 = df1.drop(columns=['mpg', 'name'])

# c. print the first few rows of the modified data frame
df1.head()
```

Out[176]:

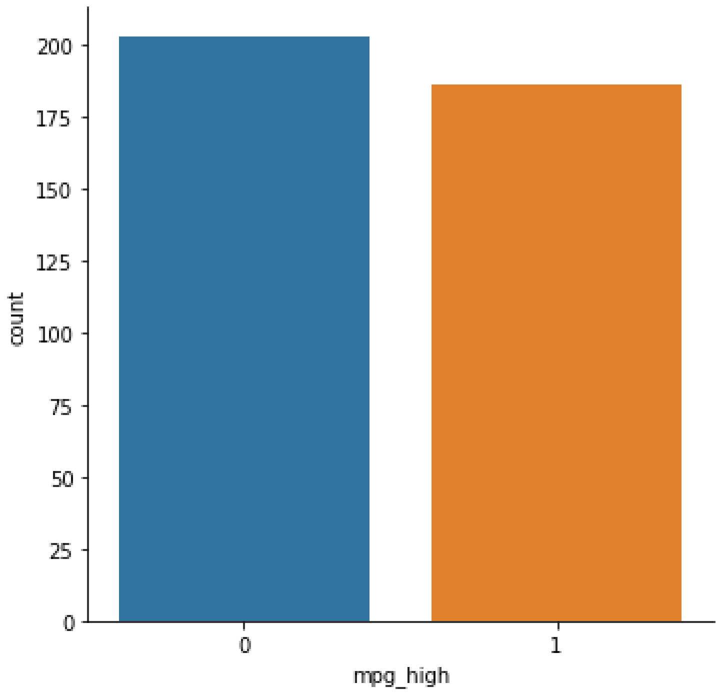
	cylinders	displacement	horsepower	weight	acceleration	year	origin	mpg_high
0	4	307.0	130	3504	12.0	70.0	1	0
1	4	350.0	165	3693	11.5	70.0	1	0
2	4	318.0	150	3436	11.0	70.0	1	0
3	4	304.0	150	3433	12.0	70.0	1	0
6	4	454.0	220	4354	9.0	70.0	1	0

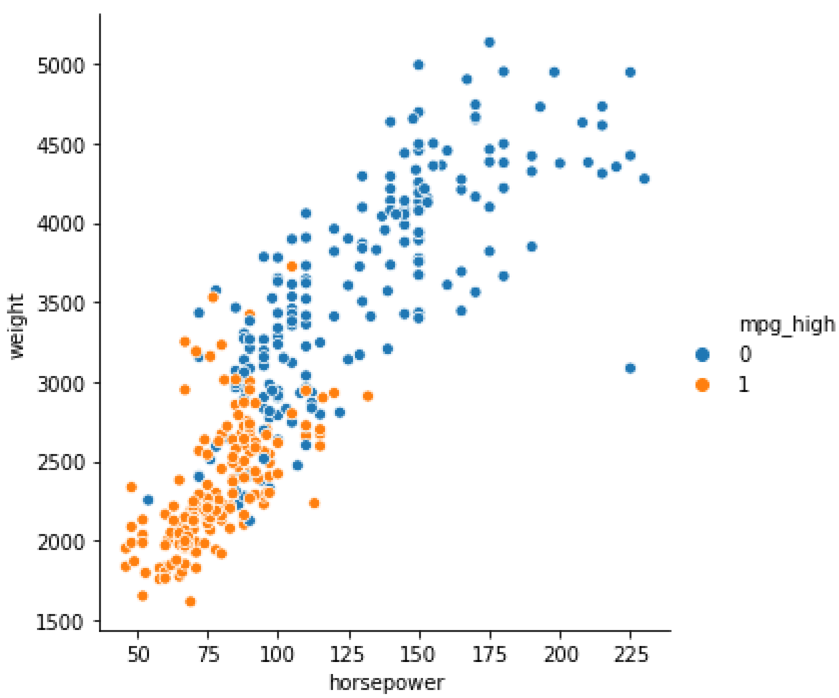
6. Data exploration with graphs (15 points)

```
a. seaborn catplot on the mpg_high column
b. seaborn relplot with horsepower on the x axis, weight on the y axis, setting hue
or style to mpg_high
c. seaborn boxplot with mpg_high on the x axis and weight on the y axis
d. for each graph, write a comment indicating one thing you learned about the data
 from the graph

Commentary:
Seaborn Catplot of mpg_high: From this graph we can see that there are more instanc
es with a mpg below the average mpg than instances that have an mpg higher than the
average

Seaborn Relplot of horsepower and weight: From this graph we learned that automobil
es that have mpg score higher than the average, tend to have a lower horsepower sco
re. In other words, automobiles with a mpg score lower than the average, and that w
eight more, tend to have a higher horspower score.

Seabor Boxplot of mpg_high and weight: From this graph we learned that automobiles
 with a higher weight tend to have lower mpg scores. Rather, lighter automobiles te
nd to have a higher mpg score.
```

In [177]:
```
import seaborn as sb

# a. seaborn catplot on the mpg_high column
sb.catplot(x='mpg_high', kind='count', data=df1)

# b. seaborn relplot with horsepower on the x axis, weight on the y axis, sett
ing hue or style to mpg_high
sb.relplot(x='horsepower', y='weight', data=df1, hue=df1.mpg_high)
```

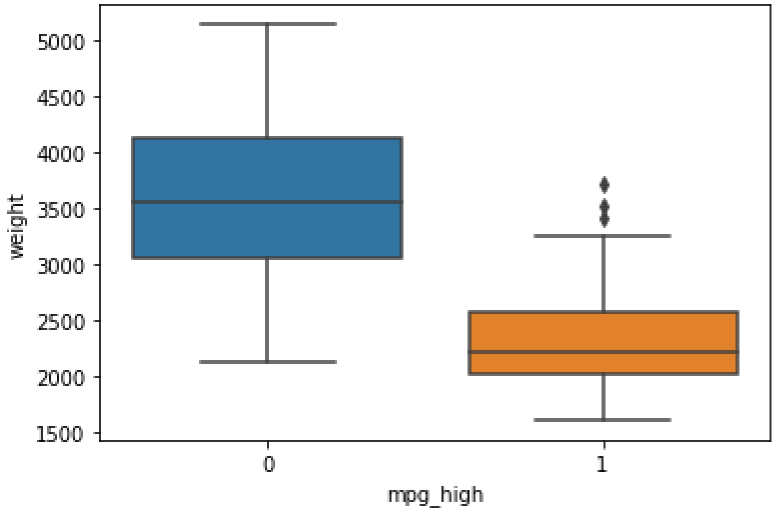
Out[177]: <seaborn.axisgrid.FacetGrid at 0x17e4f6d9888>





In [178]:
```
# c. seaborn boxplot with mpg_high on the x axis and weight on the y axis
sb.boxplot(x='mpg_high', y='weight', data=df1)
```

Out[178]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x17e4df58588>
```



7. Train/test split (5 points)

```
a. 80/20
b. use seed 1234 so we all get the same results
c. train /test X data frames consists of all remaining columns except mpg_high
d. print the dimensions of train and test
```

In [179]:
```
from sklearn.model_selection import train_test_split

# c. train /test X data frames consists of all remaining columns except mpg_hi
gh
X = df1.loc[:,['cylinders', 'displacement', 'horsepower', 'weight', 'accelerat
ion', 'year','origin']]
y = df1.mpg_high

# a. 80/20, b. use seed 1234 so we all get the same results
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando
m_state=1234)

# d. print the dimensions of train and test
print('train size:', X_train.shape)
print('test size:', X_test.shape)
```

```
train size: (311, 7)
test size: (78, 7)
```

8. Logistic Regression (10 points)

```
a. train a logistic regression model using solver lbfgs
b. test and evaluate
c. print metrics using the classification report
```

In [180]:
```
from sklearn.linear_model import LogisticRegression
import warnings
warnings.filterwarnings('ignore')

glm1 = LogisticRegression(solver='lbfgs')

# a. train a logistic regression model using solver lbfgs
glm1.fit(X_train, y_train)
glm1.score(X_train, y_train)

# b. test and evaluate
pred = glm1.predict(X_test)

# c. print metrics using the classification report
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
score, confusion_matrix
print('accuracy:', accuracy_score(y_test, pred))
print('precision:', precision_score(y_test, pred))
print('recall score:', recall_score(y_test, pred))
print('f1 score', f1_score(y_test, pred))
print('Confusion Matrix:')
confusion_matrix(y_test, pred)
```

```
accuracy: 0.8589743589743589
precision: 0.7297297297297297
recall score: 0.9642857142857143
f1 score 0.8307692307692307
Confusion Matrix:
```

Out[180]:
```
array([[40, 10],
       [ 1, 27]], dtype=int64)
```

9. Decision Tree (10 points)

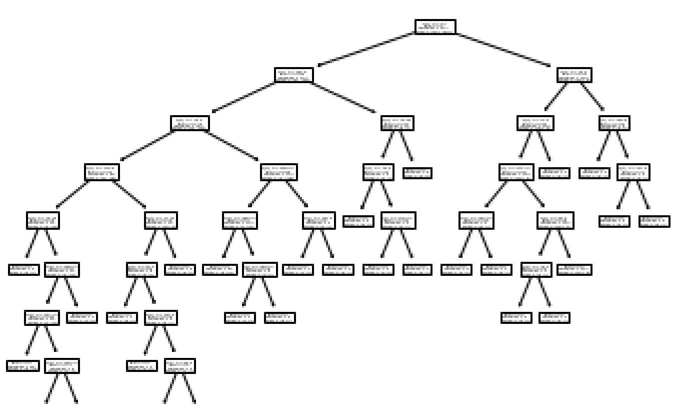
```
a. train a decision tree
b. test and evaluate
c. print the classification report metrics
d. plot the tree (optional, see: https://scikit-learn.org/stable/modules/tree.html)
```

In [181]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

# a. train a decision tree
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)

# b. test and evaluate
pred2 = dt.predict(X_test)

# c. print the classification report metrics
print('accuracy score: ', accuracy_score(y_test, pred2))
print('precision score: ', precision_score(y_test, pred2))
print('recall score: ', recall_score(y_test, pred2))
print('f1 score: ', f1_score(y_test, pred2))
print('Confusion matrix:')
print(confusion_matrix(y_test, pred2))

# d. plot the tree
tree.plot_tree(dt)
```

```
accuracy score:  0.9230769230769231
precision score:  0.8666666666666667
recall score:  0.9285714285714286
f1 score:  0.896551724137931
Confusion matrix:
[[46  4]
 [ 2 26]]
```

Out[181]:
```
[Text(215.40441176470588, 205.35999999999999, 'X[0] <= 2.5\ngini = 0.5\nsampl
es = 311\nvalue = [153, 158]'),
 Text(145.24411764705883, 181.2, 'X[2] <= 101.0\ngini = 0.239\nsamples = 173
\nvalue = [24, 149]'),
 Text(93.54705882352941, 157.04, 'X[5] <= 75.5\ngini = 0.179\nsamples = 161\n
value = [16, 145]'),
 Text(49.23529411764706, 132.88, 'X[1] <= 119.5\ngini = 0.362\nsamples = 59\n
value = [14, 45]'),
 Text(19.694117647058825, 108.72, 'X[4] <= 13.75\ngini = 0.159\nsamples = 46
\nvalue = [4, 42]'),
 Text(9.847058823529412, 84.56, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(29.541176470588237, 84.56, 'X[3] <= 2683.0\ngini = 0.087\nsamples = 44
\nvalue = [2, 42]'),
 Text(19.694117647058825, 60.400000000000006, 'X[3] <= 2377.0\ngini = 0.045\n
samples = 43\nvalue = [1, 42]'),
 Text(9.847058823529412, 36.24000000000001, 'gini = 0.0\nsamples = 38\nvalue
= [0, 38]'),
 Text(29.541176470588237, 36.24000000000001, 'X[3] <= 2385.0\ngini = 0.32\nsa
mples = 5\nvalue = [1, 4]'),
 Text(19.694117647058825, 12.079999999999984, 'gini = 0.0\nsamples = 1\nvalue
= [1, 0]'),
 Text(39.38823529411765, 12.079999999999984, 'gini = 0.0\nsamples = 4\nvalue
= [0, 4]'),
 Text(39.38823529411765, 60.400000000000006, 'gini = 0.0\nsamples = 1\nvalue
= [1, 0]'),
 Text(78.7764705882353, 108.72, 'X[4] <= 17.75\ngini = 0.355\nsamples = 13\nv
alue = [10, 3]'),
 Text(68.92941176470589, 84.56, 'X[2] <= 81.5\ngini = 0.469\nsamples = 8\nval
ue = [5, 3]'),
 Text(59.082352941176474, 60.400000000000006, 'gini = 0.0\nsamples = 2\nvalue
= [0, 2]'),
 Text(78.7764705882353, 60.400000000000006, 'X[1] <= 131.0\ngini = 0.278\nsam
ples = 6\nvalue = [5, 1]'),
 Text(68.92941176470589, 36.24000000000001, 'gini = 0.0\nsamples = 4\nvalue =
[4, 0]'),
 Text(88.62352941176471, 36.24000000000001, 'X[2] <= 86.5\ngini = 0.5\nsample
s = 2\nvalue = [1, 1]'),
 Text(78.7764705882353, 12.079999999999984, 'gini = 0.0\nsamples = 1\nvalue =
[1, 0]'),
 Text(98.47058823529412, 12.079999999999984, 'gini = 0.0\nsamples = 1\nvalue
= [0, 1]'),
 Text(88.62352941176471, 84.56, 'gini = 0.0\nsamples = 5\nvalue = [5, 0]'),
 Text(137.85882352941178, 132.88, 'X[3] <= 3250.0\ngini = 0.038\nsamples = 10
2\nvalue = [2, 100]'),
 Text(118.16470588235295, 108.72, 'X[3] <= 2880.0\ngini = 0.02\nsamples = 100
\nvalue = [1, 99]'),
 Text(108.31764705882354, 84.56, 'gini = 0.0\nsamples = 94\nvalue = [0, 9
4]'),
 Text(128.01176470588237, 84.56, 'X[3] <= 2920.0\ngini = 0.278\nsamples = 6\n
value = [1, 5]'),
 Text(118.16470588235295, 60.400000000000006, 'gini = 0.0\nsamples = 1\nvalue
= [1, 0]'),
 Text(137.85882352941178, 60.400000000000006, 'gini = 0.0\nsamples = 5\nvalue
= [0, 5]'),
 Text(157.5529411764706, 108.72, 'X[1] <= 151.5\ngini = 0.5\nsamples = 2\nval
ue = [1, 1]'),
 Text(147.7058823529412, 84.56, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
```

```
 Text(167.4, 84.56, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(196.94117647058823, 157.04, 'X[4] <= 14.45\ngini = 0.444\nsamples = 12
\nvalue = [8, 4]'),
 Text(187.09411764705882, 132.88, 'X[5] <= 76.0\ngini = 0.444\nsamples = 6\nv
alue = [2, 4]'),
 Text(177.24705882352941, 108.72, 'gini = 0.0\nsamples = 3\nvalue = [0, 3]'),
 Text(196.94117647058823, 108.72, 'X[3] <= 2760.0\ngini = 0.444\nsamples = 3
\nvalue = [2, 1]'),
 Text(187.09411764705882, 84.56, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(206.78823529411767, 84.56, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(206.78823529411767, 132.88, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]'),
 Text(285.56470588235294, 181.2, 'X[5] <= 79.5\ngini = 0.122\nsamples = 138\n
value = [129, 9]'),
 Text(265.8705882352941, 157.04, 'X[4] <= 21.6\ngini = 0.045\nsamples = 129\n
value = [126, 3]'),
 Text(256.02352941176474, 132.88, 'X[3] <= 2737.0\ngini = 0.031\nsamples = 12
8\nvalue = [126, 2]'),
 Text(236.3294117647059, 108.72, 'X[3] <= 2674.0\ngini = 0.444\nsamples = 3\n
value = [2, 1]'),
 Text(226.4823529411765, 84.56, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(246.1764705882353, 84.56, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(275.71764705882356, 108.72, 'X[2] <= 83.0\ngini = 0.016\nsamples = 125
\nvalue = [124, 1]'),
 Text(265.8705882352941, 84.56, 'X[2] <= 79.5\ngini = 0.375\nsamples = 4\nval
ue = [3, 1]'),
 Text(256.02352941176474, 60.400000000000006, 'gini = 0.0\nsamples = 3\nvalue
= [3, 0]'),
 Text(275.71764705882356, 60.400000000000006, 'gini = 0.0\nsamples = 1\nvalue
= [0, 1]'),
 Text(285.56470588235294, 84.56, 'gini = 0.0\nsamples = 121\nvalue = [121,
0]'),
 Text(275.71764705882356, 132.88, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(305.25882352941176, 157.04, 'X[1] <= 196.5\ngini = 0.444\nsamples = 9\n
value = [3, 6]'),
 Text(295.4117647058824, 132.88, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(315.1058823529412, 132.88, 'X[1] <= 247.0\ngini = 0.48\nsamples = 5\nva
lue = [3, 2]'),
 Text(305.25882352941176, 108.72, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(324.95294117647063, 108.72, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]')]
```



10. Analysis (20 points)

```
a. which algorithm performed better?
b. compare accuracy, recall and precision metrics by class
c. give your analysis of why the better-performing algorithm might have outperforme
d the other

Commentary:
The decision tree algorithm performed better than the logistic regression algorith
m. It had a higher accuracy score (92 vs 85), a higher precision score (86 vs 72),
 but a slightly smaller recall score (92.8 vs 96.6). The reason for this outcome is
most likely due to the nature of the data. Logistic Regression models will tend to
 perform better on data that is linearly separable. On the other hand, Decision Tre
es are non-linear classifiers which means they do not require the data to be linear
ly separable, allowimg them to outperform logistic regression models when the data
 is not linearly separable.
```