Importing Data

```
import pandas as pd
import numpy as np
import seaborn as sb

auto = pd.read_csv('Auto.csv')
print(auto.head(), '\n')
print('Data dimensions: ', auto.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

Data dimensions:  (392, 9)
```

Data Exploration

```
print(auto.mpg.describe(), '\n')
print(auto.weight.describe(), '\n')
print(auto.year.describe())
```

```
count    392.000000
mean      23.445918
std        7.805007
min        9.000000
25%       17.000000
50%       22.750000
75%       29.000000
max       46.600000
Name: mpg, dtype: float64

count     392.000000
mean     2977.584184
std       849.402560
min      1613.000000
25%      2225.250000
50%      2803.500000
75%      3614.750000
max      5140.000000
```

```
Name: weight, dtype: float64

count    390.000000
mean      76.010256
std        3.668093
min       70.000000
25%       73.000000
50%       76.000000
75%       79.000000
max       82.000000
Name: year, dtype: float64
```

Based on these descriptions, the average value of mpg is 23.445918 and the range is 37.6 (46.6 - 9). The average value of weight is 2977.584184 and the range is 3527 (5140 - 1613). The average value of year is 76.010256 and the range is 12 (82 - 70).

Data Cleaning

```
print(auto.dtypes, '\n')

auto['cylinders'] = auto.cylinders.astype('category').cat.codes
auto = auto.astype({'origin': 'category'})

print(auto.dtypes, '\n')
print(auto.cylinders.describe())
```

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

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

count    392.000000
mean       2.209184
std        1.331160
```

```
min        0.000000
25%        1.000000
50%        1.000000
75%        4.000000
max        4.000000
Name: cylinders, dtype: float64
```

```
print(auto.isnull().sum(), '\n')

auto = auto.dropna()
print('Data dimensions: ', auto.shape)
```

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

Data dimensions:  (389, 9)
```

```
auto['mpg_high'] = np.where(auto['mpg'] > np.mean(auto['mpg']), 1, 0)
auto = auto.astype({'mpg_high': 'category'})
auto = auto.drop(columns = ['mpg', 'name'])
print(auto.dtypes, '\n')
print(auto.head())
```

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

   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
```

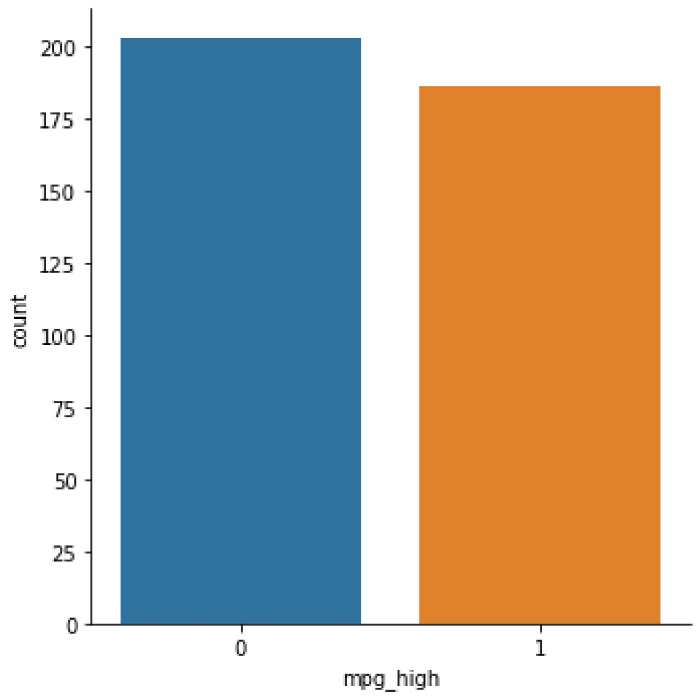
```
3          0
```

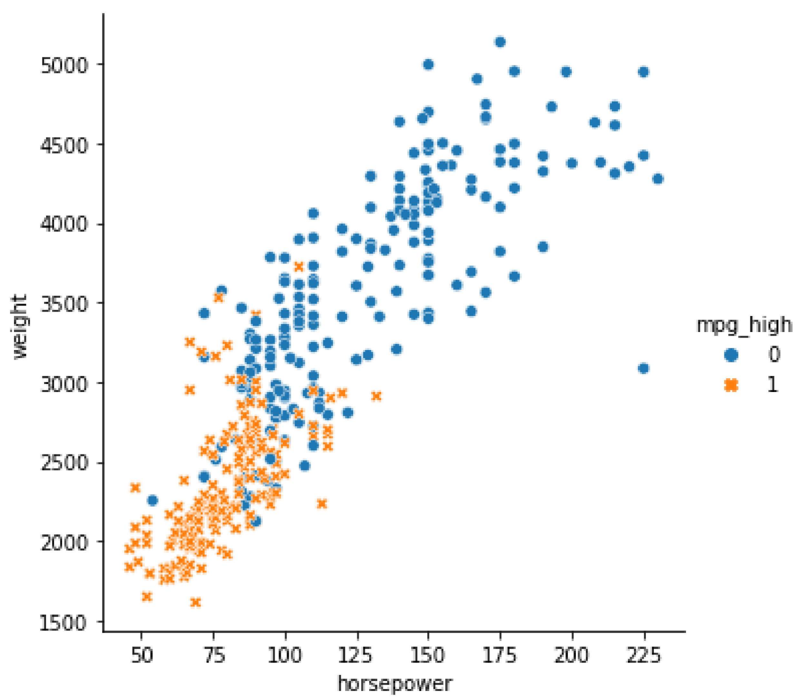
Data Plotting

```
sb.catplot(x = 'mpg_high', kind = 'count', data = auto)

sb.relplot(x = 'horsepower', y = 'weight', data = auto, hue = auto.mpg_high, style = auto.mpg
```

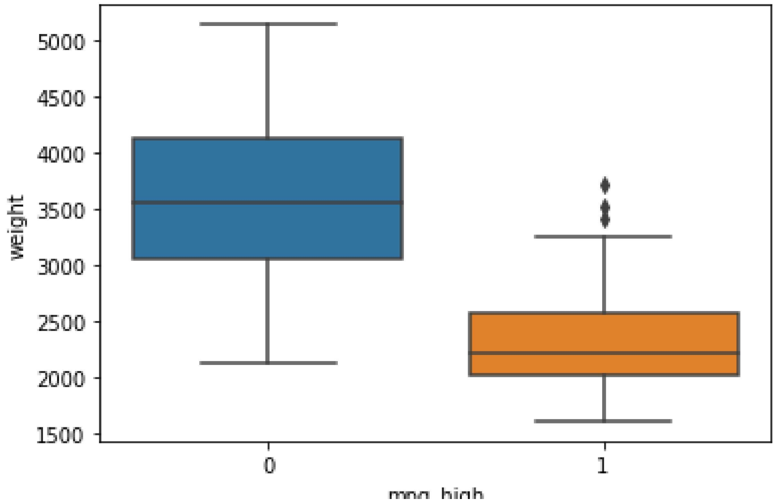
```
<seaborn.axisgrid.FacetGrid at 0x7ff798caa490>
```






```
sb.boxplot(x = 'mpg_high', y = 'weight', data = auto)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff798b21fd0>
```



From these graphs, I can see that there is a pretty even split between vehicles with below-average miles per gallon and above-average miles per gallon, with slightly more falling on the lower end. This could also be learned from the slight discrepancy between median and mean. Vehicles with above-average gas mileage tend to weigh less and have lower horesepowers than vehicles with below-average gas mileage. There is less variety in weight among vehicles with above-average gas mileage than those with below-average gas mileage, which can be seen in both the scatter plot and box plot.

```
from sklearn.model_selection import train_test_split

X = auto.iloc[:, 0:6]
Y = auto.iloc[:, 7]

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state = 12
print('Predictor train and test dimensions: ', X_train.shape, X_test.shape, '\n')
print('Target train and test dimensions: ', Y_train.shape, Y_test.shape)
```

```
Predictor train and test dimensions:  (311, 6) (78, 6)

Target train and test dimensions:  (311,) (78,)
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusic

clf = LogisticRegression(class_weight = 'balanced', solver = 'lbfgs')
clf.fit(X_train, Y_train)
print(clf.score(X_train, Y_train), '\n')
pred = clf.predict(X_test)

print('Confusion Matrix: ', confusion_matrix(Y_test, pred))
```

```
print(classification_report(Y_test, pred))
```

```
0.9035369774919614

Confusion Matrix:  [[40 10]
 [ 1 27]]
              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
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier, plot_tree

clft = DecisionTreeClassifier()
clft.fit(X_train, Y_train)
predt = clft.predict(X_test)

print('Accuracy: ', accuracy_score(Y_test, predt))
print('Precision: ', precision_score(Y_test, predt))
print('Recall: ', recall_score(Y_test, predt))
print('F1: ', f1_score(Y_test, predt))
print('Confusion Matrix: ', confusion_matrix(Y_test, predt))

text = plot_tree(clft, label = 'root')
```

```
Accuracy:  0.9102564102564102
Precision:  0.8387096774193549
```

Neural Networks

```
Confusion Matrix:  [[45  5]
```

```
from sklearn import preprocessing
from sklearn.neural_network import MLPClassifier

scale = preprocessing.StandardScaler().fit(X_train)
X_train = scale.transform(X_train)
X_test = scale.transform(X_test)

clfn1 = MLPClassifier(solver = 'adam', hidden_layer_sizes = (5, 3), max_iter = 1000, random_s
clfn1.fit(X_train, Y_train)
predn1 = clfn1.predict(X_test)
print('Confusion Matrix: ', confusion_matrix(Y_test, predn1))
print(classification_report(Y_test, predn1))
```

```
Confusion Matrix:  [[42  8]
 [ 1 27]]
              precision    recall  f1-score   support

           0       0.98      0.84      0.90        50
           1       0.77      0.96      0.86        28

    accuracy                           0.88        78
   macro avg       0.87      0.90      0.88        78
weighted avg       0.90      0.88      0.89        78
```

```
clfn2 = MLPClassifier(solver = 'lbfgs', hidden_layer_sizes = (5, 4, 3), max_iter = 1000, ranc
clfn2.fit(X_train, Y_train)
predn2 = clfn2.predict(X_test)
print('Confusion Matrix: ', confusion_matrix(Y_test, predn2))
print('Accuracy: ', accuracy_score(Y_test, predn2))
```

```
Confusion Matrix:  [[45  5]
 [ 2 26]]
Accuracy:  0.9102564102564102
```

Between these two networks, the latter performed slightly better, though both had decent results relative to the other models. Based on documentation, the lbfgs optimizer worked better with small data sets than the default gradient descent, allowing me to add another layer. The added learning from another layer provided some additional insight, though any more nodes or layers than this resulted in overfitting and a lower accuracy. Both networks had enough iterations to properly converge. Running the full classification report on the second network resulted in errors.

From the metric reports, it seemed that neural networks had the best precision, particularly for low miles per gallon, closely followed by logistic regression and decision trees in last. Decision trees, in turn, had the best recall, followed by neural networks and logistic regression in last. For overall accuracy, the larger neural network and the decision tree were tied, with the logistic regression and stochastic gradient descent neural network trailing closely behind. Decision trees and lbfgs optimization are both known to work well with small data sets like this one. Neural networks are very powerful, and while deep learning requires much more data than what is provided here, a small network was able to outperform logistic regression without overfitting. Between decision trees and logistic regression, trees have less bias than logistic regression, allowing for a slightly more accurate classification when the subdivision of the target is not so clear.

I have used Python and its libraries in the past, so I am far more comfortable using it than R, which I have only just started learning this semester. I actually used numpy and sklearn in particular in high school when I read Neural Networks and Deep Learning, one of the recommended books for this class. The tree-shaking for sklearn is easier to manage than the imports in RStudio, which had to be done manually and were not all available for download. The nesting syntax of Python is also more pleasing, as it resembles the syntax of most languages I have used. Finally, the default graphs produced by seaborn are nicer than the default graphs in R, though with some effort both can be made much cleaner.

0s completed at 5:00 PM