# Data Exploration

## Read the Auto Data

Reading the auto.csv with pandas

```
import pandas as pd
df = pd.read_csv('auto.csv')

df.head()
```

| | mpg | cylinders | displacement | horsepower | weight | acceleration | year | origin | name |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 18.0 | 8 | 307.0 | 130 | 3504 | 12.0 | 70.0 | 1 | chevrolet chevelle malibu |
| 1 | 15.0 | 8 | 350.0 | 165 | 3693 | 11.5 | 70.0 | 1 | buick skylark 320 |
| 2 | 18.0 | 8 | 318.0 | 150 | 3436 | 11.0 | 70.0 | 1 | plymouth satellite |
| 3 | 16.0 | 8 | 304.0 | 150 | 3433 | 12.0 | 70.0 | 1 | amc rebel sst |
| 4 | 17.0 | 8 | 302.0 | 140 | 3449 | NaN | 70.0 | 1 | ford torino |

Dimensions of auto.csv

```
df.shape
```

```
(392, 9)
```

## Describing the Data

```
df['mpg'].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
```

```
df['weight'].describe()
```

```
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
```

```
df['year'].describe()
```

```
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
```

## Averages and Ranges

### MPG

- Avg: 23.45
- Range: 37.6

### Weight

- Avg: 2977.58
- Range: 3527

Year

- Avg: 76.01
- Range: 12

## Checking Data Types

```
df.dtypes
```

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

Changing cylinders column to categorical

```
df.cylinders = df.cylinders.astype('category')
```

Changing origin column to categorical

```
df.origin = df.origin.astype('category').cat.codes
```

```
df.dtypes
```

```
mpg               float64
cylinders        category
```

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

## Dealing with NAs

```
df = df.dropna()

# New dimensions
df.shape
```

```
(389, 9)
```

+ Code  + Text

## Modify Columns

```
mpg_high = df.mpg
mpg_high = list(map(lambda x: 1 if x > df.mpg.mean() else 0, mpg_high))

df = df.drop(columns=['mpg', 'name'])

df = df.assign(mpg_high = mpg_high)
df.mpg_high = df.mpg_high.astype('category')

df.head()
```

| cylinders | displacement | horsepower | weight | acceleration | year | origin | mpg_high |
|---|---|---|---|---|---|---|---|

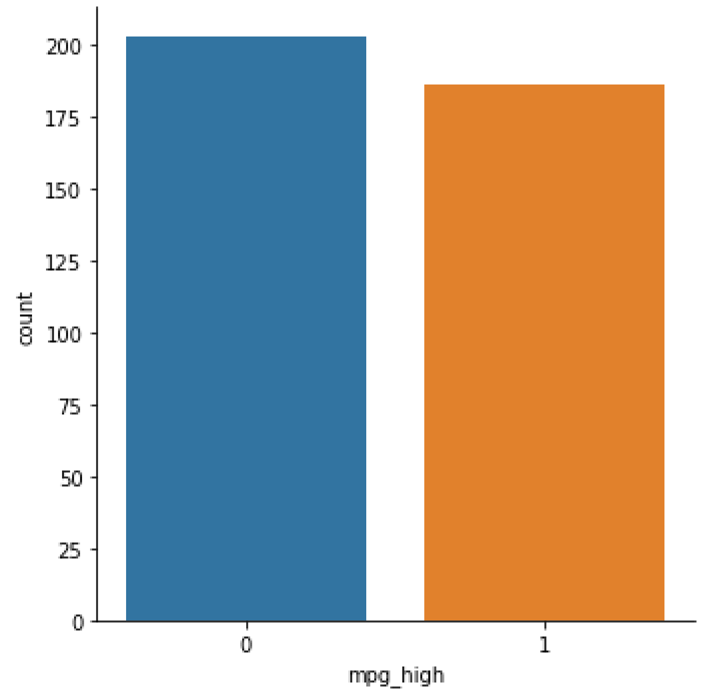
# Data Exploration with Graphs

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 2 | 8 | 318.0 | 150 | 3436 | 11.0 | 70.0 | 0 | 0 |

```
import seaborn as sb

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

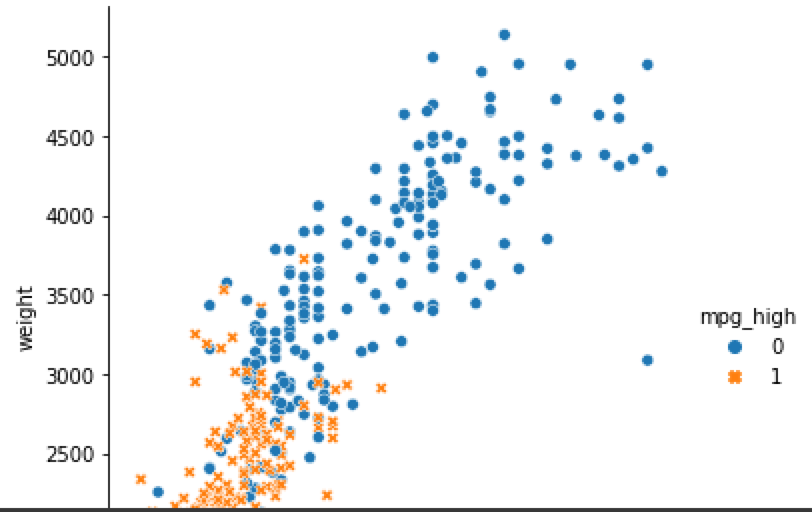
```
<seaborn.axisgrid.FacetGrid at 0x7fc52ced9610>
```




This graph shows that a little less than half the cars have a mpg above average. A little more than half are below average.

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

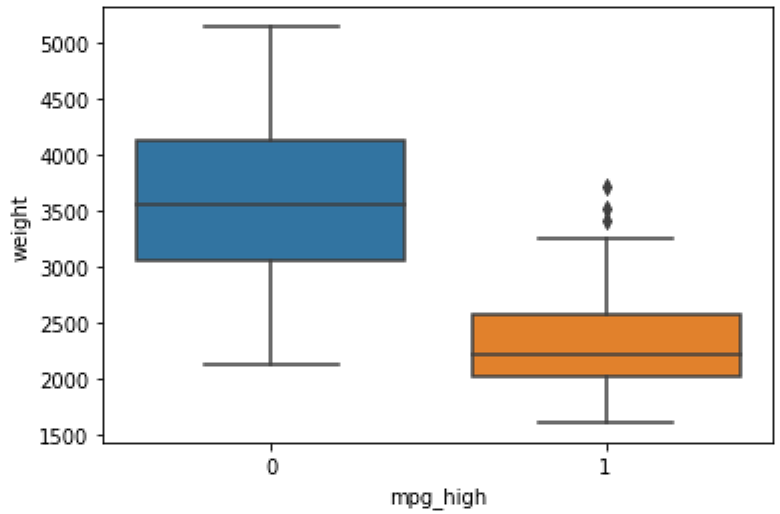
<seaborn.axisgrid.FacetGrid at 0x7fc51e279110>




It seems that cars with less weight and horsepower tend to have an mpg above average.

```
sb.boxplot(x="mpg_high", y="weight", data=df)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7fc51ddbc7d0>




This graph shows that lighter vehicles tend to have a mpg above average.

## SKLearn

# Train/test split

```
# train test split
import numpy as np
from sklearn.model_selection import train_test_split

np.random.seed(1234)

X = df.loc[:, ['cylinders', 'displacement', 'horsepower',   'weight',   'acceleration', 'year', 'origin']]
y = df.mpg_high

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1234)

print('train size:', X_train.shape)
print('test size:', X_test.shape)
```

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

# Logisitic Regression

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(max_iter=300)
clf.fit(X_train, y_train)
clf.score(X_train, y_train)
```

```
0.9067524115755627
```

```
# make predictions

pred = clf.predict(X_test)
```

```
# evaluate
from sklearn.metrics import accuracy_score
print('accuracy = ', accuracy_score(y_test, pred))
```

```
# classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))

# confusion matrix
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, pred)
```

```
accuracy =  0.8717948717948718
              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

array([[41,  9],
       [ 1, 27]])
```

## Decision Tree

```
from sklearn.tree import DecisionTreeClassifier

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

```
DecisionTreeClassifier()
```

```
# make predictions

pred = clf.predict(X_test)
```

```
# Evaluate
print('accuracy = ', accuracy_score(y_test, pred))

# Classification Report
```

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

# confusion matrix
confusion_matrix(y_test, pred)
```

```
accuracy =  0.9230769230769231
              precision    recall  f1-score   support

           0       0.96      0.92      0.94        50
           1       0.87      0.93      0.90        28

    accuracy                           0.92        78
   macro avg       0.91      0.92      0.92        78
weighted avg       0.93      0.92      0.92        78

array([[46,  4],
       [ 2, 26]])
```

## Neural Network

```
# normalize the data
from sklearn import preprocessing

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

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

```
# train
from sklearn.neural_network import MLPClassifier

clf = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(6, 2), max_iter=500, random_state=1234)
clf.fit(X_train_scaled, y_train)
```

```
MLPClassifier(hidden_layer_sizes=(6, 2), max_iter=500, random_state=1234,
              solver='lbfgs')
```

```
# make predictions

pred = clf.predict(X_test_scaled)
```

```
# output results

print('accuracy = ', accuracy_score(y_test, pred))

# Classification Report
print(classification_report(y_test, pred))

# Confusion Matrix
confusion_matrix(y_test, pred)
```

```
accuracy =  0.8974358974358975
              precision    recall  f1-score   support

           0       0.96      0.88      0.92        50
           1       0.81      0.93      0.87        28

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

array([[44,  6],
       [ 2, 26]])
```

```
# try different settings

clf = MLPClassifier(solver='sgd', hidden_layer_sizes=(2,), max_iter=1500, random_state=1234)
clf.fit(X_train_scaled, y_train)
```

```
MLPClassifier(hidden_layer_sizes=(2,), max_iter=1500, random_state=1234,
              solver='sgd')
```

```
# make predictions

pred = clf.predict(X_test_scaled)
```

```
# output results

print('accuracy = ', accuracy_score(y_test, pred))

# Classification Report
```

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

# Confusion Matrix
confusion_matrix(y_test, pred)
```

```
accuracy =  0.8974358974358975
              precision    recall  f1-score   support

           0       1.00      0.84      0.91        50
           1       0.78      1.00      0.88        28

    accuracy                           0.90        78
   macro avg       0.89      0.92      0.89        78
weighted avg       0.92      0.90      0.90        78

array([[42,  8],
       [ 0, 28]])
```

### Analysis of the 2 Neural Networks

After playing around with the settings on both neural networks, the best I could achieve was an accuracy of .90 for both of them. This is a really high accuracy and I think the performance of the models was similar because they're uncovering the same underlying patterns in the data with the most accuracy possible. If .90 is near the highest accuracy achievable, both models arrived to the same conclusion but on different paths.

## Analysis

Suprisingly Decision Tree outperformed all the other algorithms.

### Logisitic Regression

- Accuracy: .87
- Recall: .96
- Precision: .75

Generally underperformed a bit compared to the other algorithms

### Decision Tree

- Accuracy: .92
- Recall: .93

- Precision: .87

Generally the best algorithm and it had the highest precision and accuracy.

### Neural Network lbfgs

- Accuracy: .90
- Recall: .93
- Precision: .81

Did well and was the middle of the pack in all metrics.

### Neural Network sgd

- Accuracy: .90
- Recall: 1
- Precision: .78

Did well and interestingly had perfect recall.

## Why did DT perform the best?

I believe DT performed the best because the data was very linearly seperable. This was clear when we were exploring the data. I expected the neural networks to outperform it, but they may have overfit in a few small places.

## R vs SKLearn

I'm happy that I used R to learn ML. The code itself is shows more of how the algorithm works than SKLearn. That being said, I vastly prefer Python as a language and SKLearn is generally easier to use than R. The ML skills were pretty transferable. Although, I like the way R Notebooks are setup more than Google Colab. Colab is more tedious in my opinion.