In [ ]:
```
from google.colab import files
uploaded = files.upload()
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Auto.csv to Auto (8).csv
```

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

df = pd.read_csv('Auto.csv')
print(df.head)
print('\nDimesnions of data frame:',df.shape)
```

```
<bound method NDFrame.head of       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
..    ...        ...           ...         ...     ...           ...   ...
387  27.0          4         140.0          86    2790          15.6  82.0
388  44.0          4          97.0          52    2130          24.6  82.0
389  32.0          4         135.0          84    2295          11.6  82.0
390  28.0          4         120.0          79    2625          18.6  82.0
391  31.0          4         119.0          82    2720          19.4  82.0

     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
..      ...                        ...
387       1           ford mustang gl
388       2                 vw pickup
389       1             dodge rampage
390       1               ford ranger
391       1                chevy s-10

[392 rows x 9 columns]>

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

In [ ]:
```
print(df.mpg.describe())
print(df.weight.describe())
print(df.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
```

The average of MPG is 23.44 MPG and the range is from 9 MPG to 46.6 MPG.

The average of Weight is 2977.58 LB and the range is from 1613 LB to 5140 LB.

The average for Year is 76.01 Years while the range is from 70 Years to 82 Years.

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

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

In [ ]:
```
df1 = df.copy()
df1.cylinders = df1.cylinders.astype('category').cat.codes
df1.origin = df1.origin.astype('category')
print(df1.dtypes)
```

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

In [ ]:
```
df1.isnull().sum()
df1 = df1.dropna()
```

In [ ]:
```
print('\nDimensions of data frame:', df1.shape)
```

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

In [ ]:
```
import numpy as np
df1['mpg_high'] = np.where(df1.mpg > np.mean(df1.mpg), 1, 0)
df1.drop('mpg', inplace=True, axis=1)
df1.drop('name', inplace=True, axis=1)
```

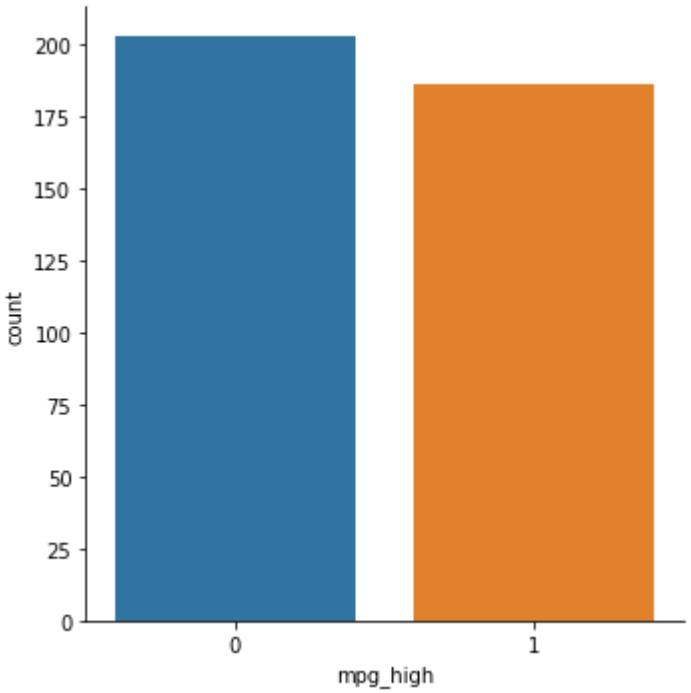
In [ ]:
```
print(df1.head())
```

```
   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
6         0
```

In [ ]:
```
import seaborn as sb
sb.catplot(x="mpg_high", kind='count',data=df1)
```

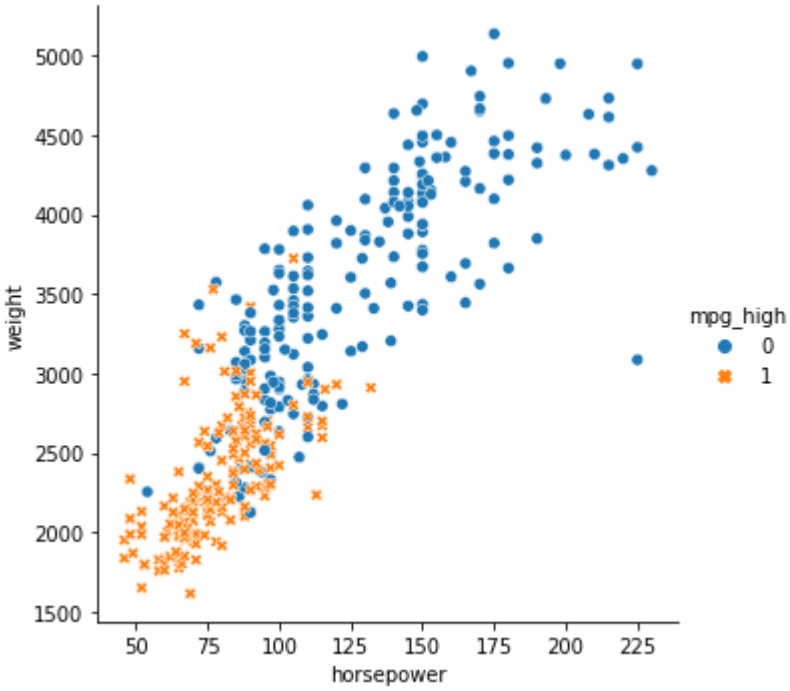
Out[ ]: <seaborn.axisgrid.FacetGrid at 0x7fdcf9dfa990>



There is more cars that are under the average MPG than over the average MPG.

In [ ]:
```
sb.relplot(x="horsepower", y='weight', data=df1, hue=df1.mpg_high, style=df1.mpg_high)
```

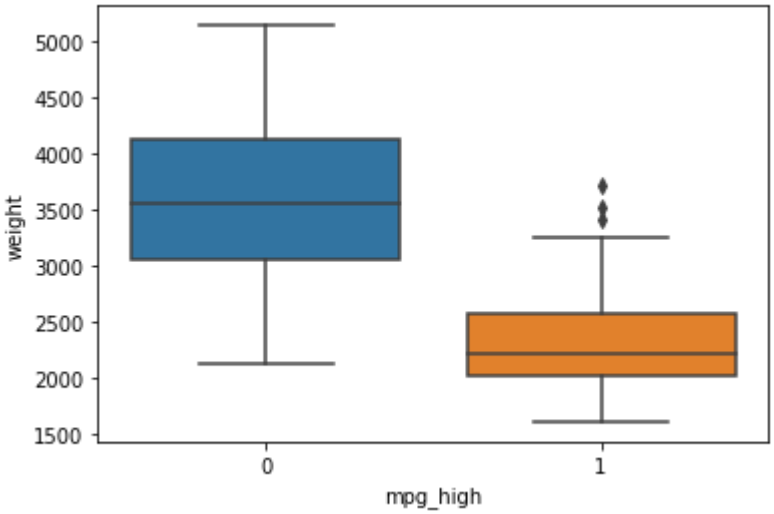
Out[ ]: <seaborn.axisgrid.FacetGrid at 0x7fdcf9304310>



The cars that have less MPG compared to the average MPG have higher horsepower and tend to weigh heavier.

In [ ]:
```
sb.boxplot(x='mpg_high',y='weight',data=df1)
```

Out[ ]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fdcf91268d0>
```



There is some correlation between the cars with lower than average MPG and how much they weigh. The cars with lower MPG on average weigh more than cars with a higher MPG then the average.

Splitting data into Test and Train

In [ ]:
```
from sklearn.model_selection import train_test_split
X = df1.loc[:,['cylinders','origin','displacement','horsepower','weight','acce
leration', 'year']]
Y = df1.mpg_high
train_X, test_X, train_Y, test_Y = train_test_split(X, Y, test_size=0.2,random
_state=1234)

print('\ntrain_Y:',train_X.shape)
print('\ntest_Y:',test_X.shape)
```

```
train_Y: (311, 7)

test_Y: (78, 7)
```

Logistic Regression

In [ ]:
```
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression(max_iter=400)
LR.fit(train_X,train_Y)
LR.score(train_X, train_Y)
```

Out[ ]: 0.9035369774919614

In [ ]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
score
LRPred = LR.predict(test_X)

print('accuracy score: ', accuracy_score(test_Y, LRPred))
print('precision score: ', precision_score(test_Y, LRPred))
print('recall score: ', recall_score(test_Y, LRPred))
print('f1 score: ', f1_score(test_Y, LRPred))
```

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

Decision Tree

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

DT = DecisionTreeClassifier()
DT.fit(train_X, train_Y)

DTPred = DT.predict(test_X)
```

In [ ]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_
score

print('accuracy score: ', accuracy_score(test_Y, DTPred))
print('precision score: ', precision_score(test_Y, DTPred))
print('recall score: ', recall_score(test_Y, DTPred))
print('f1 score: ', f1_score(test_Y, DTPred))
```

```
accuracy score:  0.9358974358974359
precision score:  0.8709677419354839
recall score:  0.9642857142857143
f1 score:  0.9152542372881356
```

In [ ]:
```
from sklearn import tree
tree.plot_tree(DT)
```

Neural Network

In [ ]:
```
from sklearn import preprocessing

scaler = preprocessing.StandardScaler().fit(train_X)
X_train_scaled = scaler.transform(train_X)
X_test_scaled = scaler.transform(test_X)
```

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

NN = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5, 2), max_iter=500, ra
ndom_state=1234)
NN.fit(X_train_scaled, train_Y)
```

Out[ ]:
```
MLPClassifier(hidden_layer_sizes=(5, 2), max_iter=500, random_state=1234,
              solver='lbfgs')
```

In [ ]:
```
pred = NN.predict(X_test_scaled)
```

In [ ]:
```
print('accuracy = ', accuracy_score(test_Y, pred))

confusion_matrix(test_Y, pred)
```

```
accuracy =  0.8717948717948718
```

Out[ ]:
```
array([[43,  7],
       [ 3, 25]])
```

In [ ]:
```
from sklearn.metrics import classification_report
print(classification_report(test_Y, 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
```

Second Neural Network

In [ ]:
```
NN2 = MLPClassifier(solver='adam', hidden_layer_sizes=(10,), max_iter=1000, ra
ndom_state=1234)
NN2.fit(X_train_scaled, train_Y)
```

Out[ ]:
```
MLPClassifier(hidden_layer_sizes=(10,), max_iter=1000, random_state=1234)
```

In [ ]:
```
from sklearn.metrics._plot.confusion_matrix import confusion_matrix
NN2Pred = NN2.predict(X_test_scaled)
print('accuracy = ', accuracy_score(test_Y, NN2Pred))
confusion_matrix(test_Y,NN2Pred)
```

```
accuracy =  0.8846153846153846
```

Out[ ]:
```
array([[41,  9],
       [ 0, 28]])
```

After doing both the Neural Networks, the accuracy between two are similiar because the changes in the classifier was not enough to cause a dramatic change.

Analysis

Out of the 3 different methods, the Decision tree had the best accuracy, however, it also had the worst because it has different variation based on the path that it chose.

In [ ]:
```
print('Logistic Regression Classification Report')
print(classification_report(test_Y, LRPred))

print('Decision Tree Classification Report')
print(classification_report(test_Y,DTPred))

print('Neural Network Classification Report')
print(classification_report(test_Y,NN2Pred))
```

```
Logistic Regression Classification Report
              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

Decision Tree Classification Report
              precision    recall  f1-score   support

           0       0.98      0.92      0.95        50
           1       0.87      0.96      0.92        28

    accuracy                           0.94        78
   macro avg       0.92      0.94      0.93        78
weighted avg       0.94      0.94      0.94        78

Neural Network Classification Report
              precision    recall  f1-score   support

           0       1.00      0.82      0.90        50
           1       0.76      1.00      0.86        28

    accuracy                           0.88        78
   macro avg       0.88      0.91      0.88        78
weighted avg       0.91      0.88      0.89        78
```

The differences between Decision Tree and the rest that caused it to have a higher accuracy was the fact that it has higher scores overall compared to the other results. The f1-score was higher for classifaction therefore it was able to achieve a higher accuracy.

I highly prefer sklearn because it is easier to do things like logistic regression or decision trees because it has a function already built in and makes filling in the missing fields easier whereas if I did this in R I would have to make sure I import the right libraries and then make sure the data fits and doesn't cause errors when implementing it.