Part 1: Read Auto data

```
import pandas as pd
df = pd.read_csv('Auto.csv')
print(df.head(10))
print('Shape:',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
5  15.0          8         429.0         198    4341          10.0   NaN
6  14.0          8         454.0         220    4354           9.0  70.0
7  14.0          8         440.0         215    4312           8.5  70.0
8  14.0          8         455.0         225    4425          10.0  70.0
9  15.0          8         390.0         190    3850           8.5  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
5       1           ford galaxie 500
6       1           chevrolet impala
7       1          plymouth fury iii
8       1           pontiac catalina
9       1         amc ambassador dpl
Shape: (392, 9)
```

Part 2: Data Exploration

```
df[['mpg', 'weight', 'year']].describe()
```

	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

```
print(df.describe())
```

```
              mpg   cylinders  displacement  horsepower       weight  \
count  392.000000  392.000000    392.000000  392.000000   392.000000
mean    23.445918    5.471939    194.411990  104.469388  2977.584184
std      7.805007    1.705783    104.644004   38.491160   849.402560
min      9.000000    3.000000     68.000000   46.000000  1613.000000
25%     17.000000    4.000000    105.000000   75.000000  2225.250000
50%     22.750000    4.000000    151.000000   93.500000  2803.500000
75%     29.000000    8.000000    275.750000  126.000000  3614.750000
max     46.600000    8.000000    455.000000  230.000000  5140.000000

       acceleration        year      origin
count    391.000000  390.000000  392.000000
mean      15.554220   76.010256    1.576531
std        2.750548    3.668093    0.805518
min        8.000000   70.000000    1.000000
25%       13.800000   73.000000    1.000000
50%       15.500000   76.000000    1.000000
75%       17.050000   79.000000    2.000000
max       24.800000   82.000000    3.000000
```

- Mean and Range of MPG: 23.445981 and 9 to 46.6
- Mean and Range of Cylinders: 5.471939 and 3 to 8
- Mean and Range of Displacement: 194.411990 and 68 to 455
- Mean and Range of Horsepower: 104.469388 and 46 to 230
- Mean and Range of weight: 2977.584184 and 1613 to 5140
- Mean and Range of Acceleration: 15.554220 and 8 to 24.8
- Mean and Range of Year: 76.010256 and 70 to 82
- Mean and Range of Origin: 1.576531 and 1 to 3

Part 3: Explore Data Types

```
print(df.dtypes, '\n')
df.cylinders = df.cylinders.astype('category').cat.codes #change cylinders to categorical (use cat.code
df.origin = df.origin.astype('category') #change origin to categorical (without cat.codes)
print(df.dtypes)
```

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

mpg              float64
cylinders           int8
displacement     float64
```

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

Part 4: Deal with NAs

```
df = df.dropna() #a. delete rows with NAs
print('Dimension after dropping NAs:',df.shape)
```

```
Dimension after dropping NAs: (389, 9)
```

Part 5: Modify Columns

```
avg = df['mpg'].mean()
df2 = df.loc[df.mpg>1].copy()
df2.loc[:, 'mpg_high'] = [                    #make a new column, mpg_high, and make it categorical:
    0 if x < avg else 1 for x in df2['mpg']]  #the column == 1 if mpg > average mpg, else == 0
df2.mpg_high = df2.mpg_high.astype('category')
df2 = df2.drop(columns=['mpg', 'name'])#b. delete the mpg and name columns
df2.head(10) #c. output the first few rows of the modified data frame
```

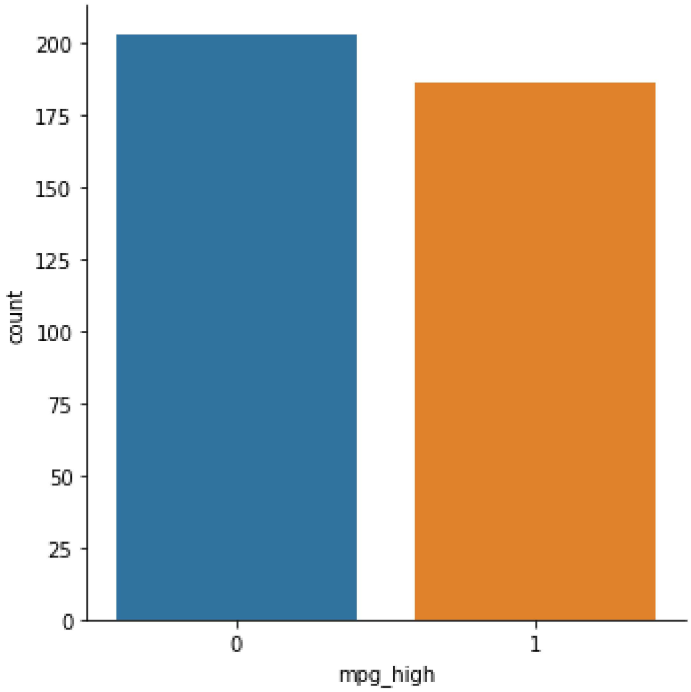
	cylinders	displacement	horsepower	weight	acceleration	year	origin	mpg_high
0	4	307.0	130	3504	12.0	70.0	1	0
1	4	350.0	165	3693	11.5	70.0	1	0
2	4	318.0	150	3436	11.0	70.0	1	0
3	4	304.0	150	3433	12.0	70.0	1	0
6	4	454.0	220	4354	9.0	70.0	1	0
7	4	440.0	215	4312	8.5	70.0	1	0
8	4	455.0	225	4425	10.0	70.0	1	0
9	4	390.0	190	3850	8.5	70.0	1	0
10	4	383.0	170	3563	10.0	70.0	1	0
11	4	340.0	160	3609	8.0	70.0	1	0

Part 6: Data Exploration with Graphs

```
import seaborn as sb
```

```
#a. seaborn catplot on the mpg_high column
sb.catplot(x = "mpg_high", kind = 'count', data = df2)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f9827bda910>
```

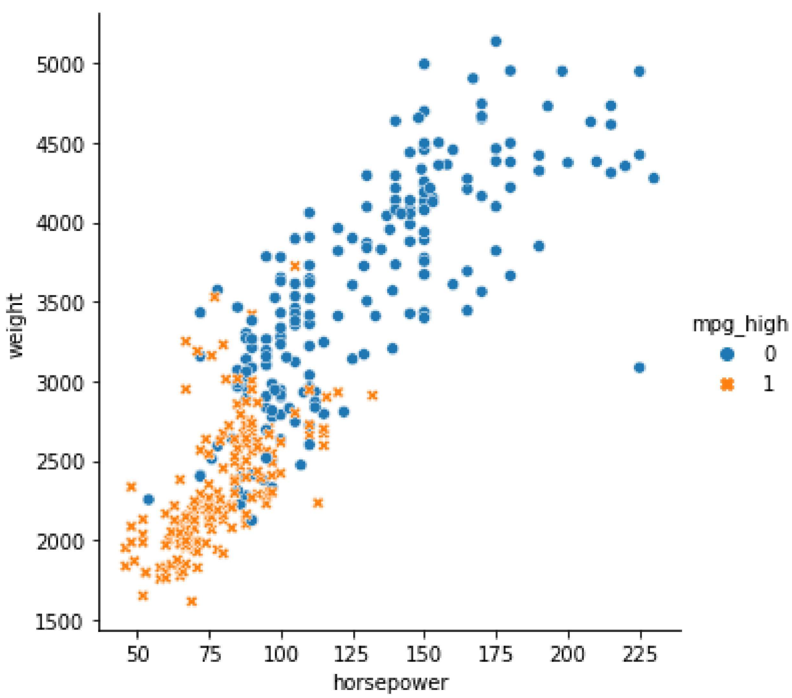


Earlier, we set mpg_high = 1 for rows that had a higher mpg value than our mean mpg value.

Hence the count of cars with mpg below average and above average is not too far apart, but there are more cars with lower mpg

```
#b. seaborn relplot with horsepower on the x axis, weight on the y axis, setting hue or style to mpg_hi
sb.relplot(x = "horsepower", y = 'weight', data = df2, hue = df2.mpg_high, style = df2.mpg_high)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f9833f1f690>
```



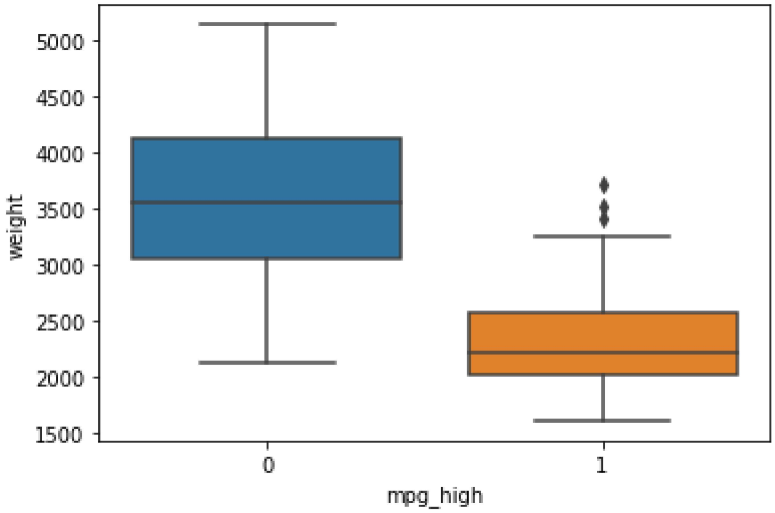
Here we can see that cars with more weight are more likely to have a higher horsepower.

Also, as we increase horsepower and weight, more cars will have less miles per gallon.

We can infer that it is because heavy cars need more gas to run.

```
#c. seaborn boxplot with mpg_high on the x axis and weight on the y axis
sb.boxplot(x = "mpg_high", y = "weight", data = df2)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9824e26610>
```



We can see that there are few outliers on the high mpg cars that have high weight values

And we can also see that the box for high mpg cars are roughly 1000 weight units below the low mpg cars.

Part 7: Train / Test split

```
from sklearn.model_selection import train_test_split

#c. train /test X data frames consists of all remaining columns except mpg_high
X = df2.iloc[:, 0:7]
y = df2.iloc[:, 7]

#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, random_state=1234)

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

```
X_train: (311, 7)
y_train: (311,)
X_test: (78, 7)
y_test: (78,)
```

Part 8: Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score, precision_score, recall_score, f1_sc

#a. train a logistic regression model using solver lbfgs
clf = LogisticRegression(solver = 'lbfgs', max_iter = 400)
clf.fit(X_train, y_train)

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

#c. print metrics using the classification report
print('Accuracy:', accuracy_score(y_test, pred))
print(classification_report(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
```

Part 9: Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from matplotlib import pyplot as plt

#a. train a decision tree
clf2 = DecisionTreeClassifier(random_state = 1234)
clf2.fit(X_train, y_train)

#b. test and evaluate
pred2 = clf2.predict(X_test)

#c. print the classification report metrics
print('Accuracy:', accuracy_score(y_test, pred2))
print(classification_report(y_test, pred2))
```

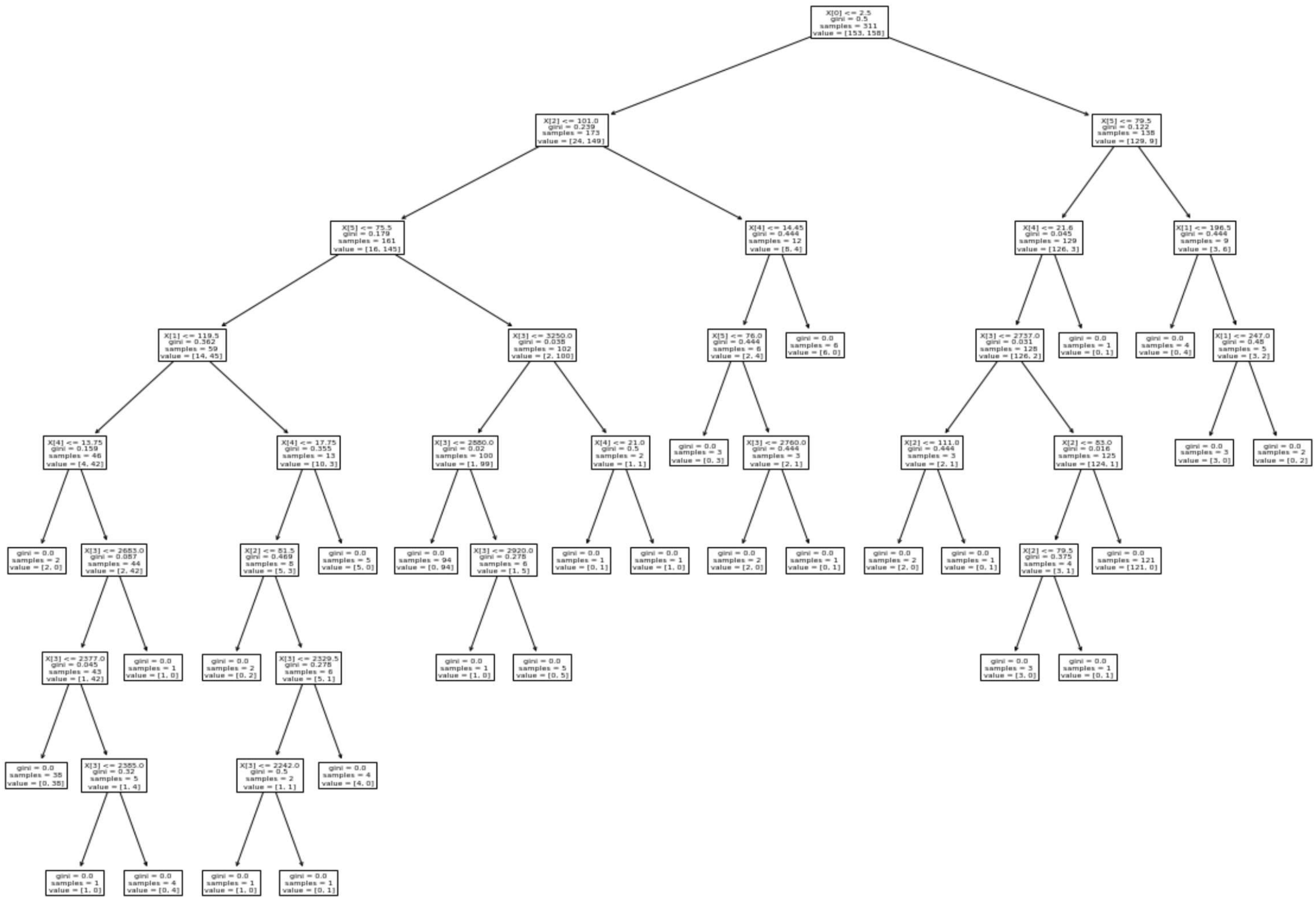
```
Accuracy: 0.9230769230769231
              precision    recall  f1-score   support

           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
```

```
#d. plot the tree (optional, see: https://scikit-learn.org/stable/modules/tree.html)
```

```
plt.figure(figsize = (20,15))
tree.plot_tree(clf2)
plt.show()
print('\n')
```



Part 10: Neural Network

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

scaler = preprocessing.StandardScaler().fit(X_train)
X_train2 = scaler.transform(X_train)
X_test2 = scaler.transform(X_test)

#a. train a neural network, choosing a network topology of your choice
clf3 = MLPClassifier(solver = 'lbfgs', hidden_layer_sizes = 6, max_iter = 400, random_state = 1234)
clf3.fit(X_train2, y_train)

#b. test and evaluate
pred3 = clf3.predict(X_test2)
print('Accuracy:', accuracy_score(y_test, pred3))
print(classification_report(y_test, pred3))
```

```
Accuracy: 0.8974358974358975
              precision    recall  f1-score   support

           0       0.94      0.90      0.92        50
           1       0.83      0.89      0.86        28

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

```
#c. train a second network with a different topology and different settings
clf4 = MLPClassifier(solver = 'lbfgs', hidden_layer_sizes = (3,3), max_iter = 400, random_state = 1234)
clf4.fit(X_train2, y_train)

#d. test and evaluate
pred4 = clf4.predict(X_test2)
print('Accuracy: ', accuracy_score(y_test, pred4))
print(classification_report(y_test, pred4))
```

```
Accuracy:  0.9102564102564102
              precision    recall  f1-score   support

           0       0.94      0.92      0.93        50
           1       0.86      0.89      0.88        28

    accuracy                           0.91        78
   macro avg       0.90      0.91      0.90        78
weighted avg       0.91      0.91      0.91        78
```

Both models performed similarly, but the second model had 1 percent point increase in accuracy. This could be because the dataset is already easy enough where adding more hidden layers won't affect the model

Part 11: Analysis

A. Which Algorithm performed better?

- The Decision Tree Model Performed with the best accuracy, recall, and precision metrics.

B. Compare Accuracy, Recall, and Precision metrics by class

- Logistic Regression : 0.897 Accuracy, 0.90 Recall, 0.92 Precision
- Decision Tree : 0.923 Accuracy, 0.92 Recall, 0.93 Precision
- Neural Network 1 : 0.897 Accuracy, 0.90 Recall, 0.90 Precision
- Neural Network 2 : 0.910 Accuracy, 0.91 Recall, 0.91 Precision

C. Analyze why some algorithms outperformed the other

Based on what we've learned so far, Neural network should be performing the best but here we can see that the decision tree showed best performance. Neural network algorithms performed well because of fine tuning that can give more layers, but it may have been overfit to the training data that caused some drop in accuracy when testing. Also according to our plot from earlier, since we are trying to classify between low and high mpg, the decision tree could have performed better since it didn't require branch pruning, and it was able to consider relationship between the variables.

D. Compare experiences using R vs. SKLearn

Since both langauges use a notebook style, it is easy to navigate between blocks of code, yet the problem exists of where if I change a variable in one block, it can change for the previous one too. Hence I tried to name every clf and pred values differently here. I think in terms of R vs. SKLearn, I think I like using SKLearn because I can work in Google Colab, a online platform that doesn't require me to download anything. From personal experience, R had given me some issues when trying to download packages, and the 'knitting to pdf' experience isn't as smooth as 'print to pdf' on google.