In [109…

```
# Author - Subash Chandra
# CS 4375 Machine Learning - Sklearn

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

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from matplotlib import pyplot as plt
from sklearn import tree
from sklearn.tree  import DecisionTreeClassifier
```

In [109…

```
# 1. Read the Auto Data
```

In [109…

```
df = pd.read_csv('Auto.csv')
df.head()
```

Out[109…

	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

In [109…

```
df.size
```

Out[109…

```
3528
```

In [109…

```
df.shape
```

Out[109…

```
(392, 9)
```

In [109…

```
# 2. Data exploration
```

In [109…

```
df.mpg.describe()
# avg weight = 23.4
# range = 9-46
```

Out[109…

```
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
```

In [109…
```
df.weight.describe()
# avg weight = 2977.58
# range = 1613-5140
```

Out[109…
```
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
```

In [109…
```
df.year.describe()
# avg weight = 76
# range = 70-82
```

Out[109…
```
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
```

In [110…
```
# 3. Explore Data Types
```

In [110…
```
df.dtypes
```

Out[110…
```
mpg             float64
cylinders         int64
displacement    float64
horsepower        int64
weight            int64
acceleration    float64
year            float64
origin            int64
name             object
dtype: object
```

In [110…
```
df.cylinders = df.cylinders.astype('category')
```

In [110…
```
df.origin = df.origin.astype('category')
```

In [110…
```
df.dtypes
```

Out[110…
```
mpg              float64
cylinders       category
displacement     float64
horsepower         int64
weight             int64
acceleration     float64
year             float64
origin          category
name              object
dtype: object
```

In [110…
```
# 4. Deal with NA's
```

In [110…
```
df.dropna(how='any',inplace=True)
df.size
```

Out[110…
```
3501
```

In [110…
```
# 5. Modify Columns
```

In [110…
```
df['mpg_high'] = np.where(df.mpg > 23.445, 1, 0)
```

In [110…
```
df.drop('mpg',inplace=True,axis=1)
```

In [111…
```
df.drop('name',inplace=True,axis=1)
```

In [111…
```
df.head()
```

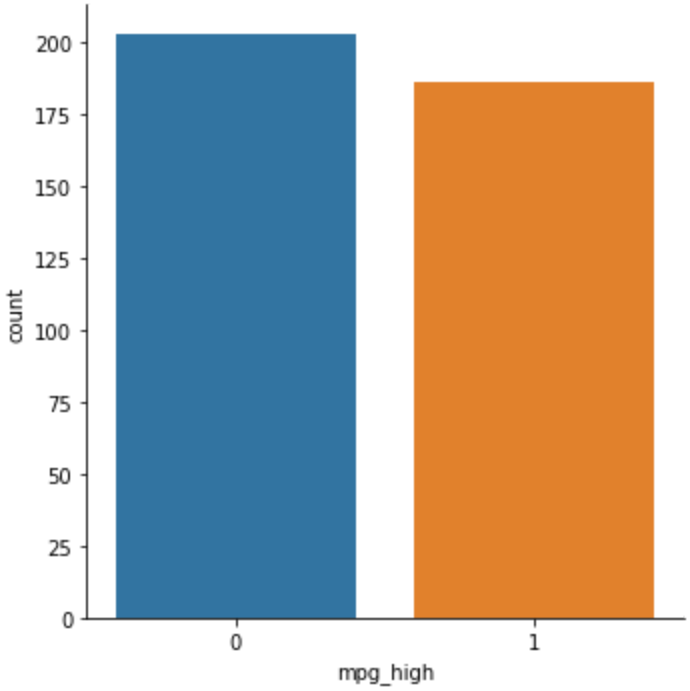
Out[111…

	cylinders	displacement	horsepower	weight	acceleration	year	origin	mpg_high
0	8	307.0	130	3504	12.0	70.0	1	0
1	8	350.0	165	3693	11.5	70.0	1	0
2	8	318.0	150	3436	11.0	70.0	1	0
3	8	304.0	150	3433	12.0	70.0	1	0
6	8	454.0	220	4354	9.0	70.0	1	0

In [111…
```
# 6.Data Exploration
```

In [111…
```
sb.catplot(x="mpg_high",kind='count',data=df)
#the distribution of cars is about equal, even though there are many more mpg_low than mpg
```

Out[111…
```
<seaborn.axisgrid.FacetGrid at 0x1de3fb5f5e0>
```

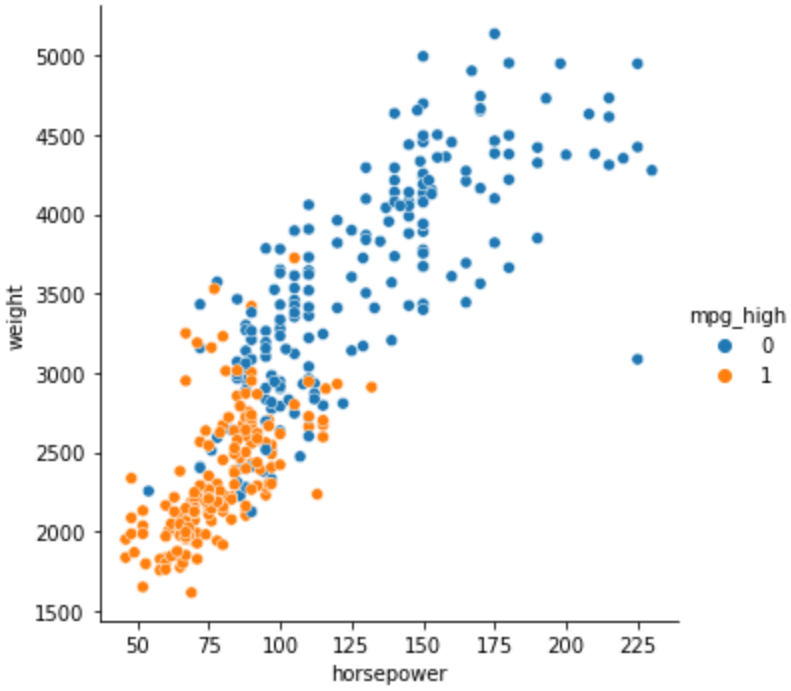


In [111…

```
sb.relplot(x='horsepower',y='weight',data=df,hue=df.mpg_high)
# The weight of the vehicle is directly proportional to the horsepower of that vehicle.
```

Out[111…

```
<seaborn.axisgrid.FacetGrid at 0x1de3d7f7970>
```

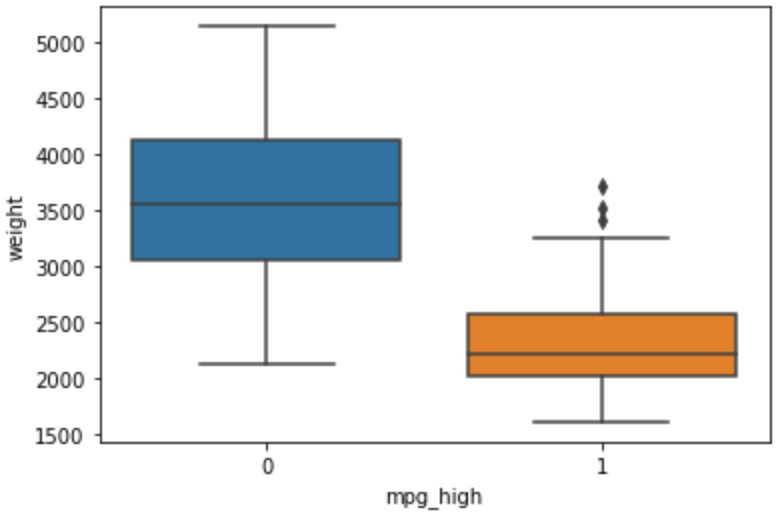


In [111…

```
sb.boxplot(x='mpg_high',y='weight',data=df)
# Although the ranges show high overlap, and cars with high MPG are much more widespread
```

Out[111…

```
<AxesSubplot:xlabel='mpg_high', ylabel='weight'>
```



In [111…
```
# 7.Train/test split
```

In [111…
```
X = df.loc[:,df.columns != 'mpg_high']
Y = df.mpg_high
```

In [111…
```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.2, random_state=1234)
```

In [111…
```
X_train.shape
```

Out[111…
```
(311, 7)
```

In [112…
```
X_test.shape
```

Out[112…
```
(78, 7)
```

In [112…
```
Y_train.shape
```

Out[112…
```
(311,)
```

In [112…
```
Y_test.shape
```

Out[112…
```
(78,)
```

In [112…
```
# 8. Logisic Regression
```

In [112…
```
clf = LogisticRegression(solver='lbfgs',max_iter=1500)
```

In [112…
```
clf.fit(X_train,Y_train)
```

Out[112…
```
LogisticRegression(max_iter=1500)
```

In [112…
```
clf.score(X_train,Y_train)
```

Out[112… 0.9003215434083601

In [112…
```
pred = clf.predict(X_test)
```

In [112…
```
print(classification_report(Y_test,pred))

# 0 = Low mpg
# 1 = High mpg
```

```
              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
```

In [112…
```
# 9. Decision Trees
```

In [113…
```
clf2 = DecisionTreeClassifier()
clf2.fit(X_train,Y_train)
clf2.score(X_train,Y_train)
```

Out[113… 1.0

In [113…
```
pred2 = clf2.predict(X_test)
```

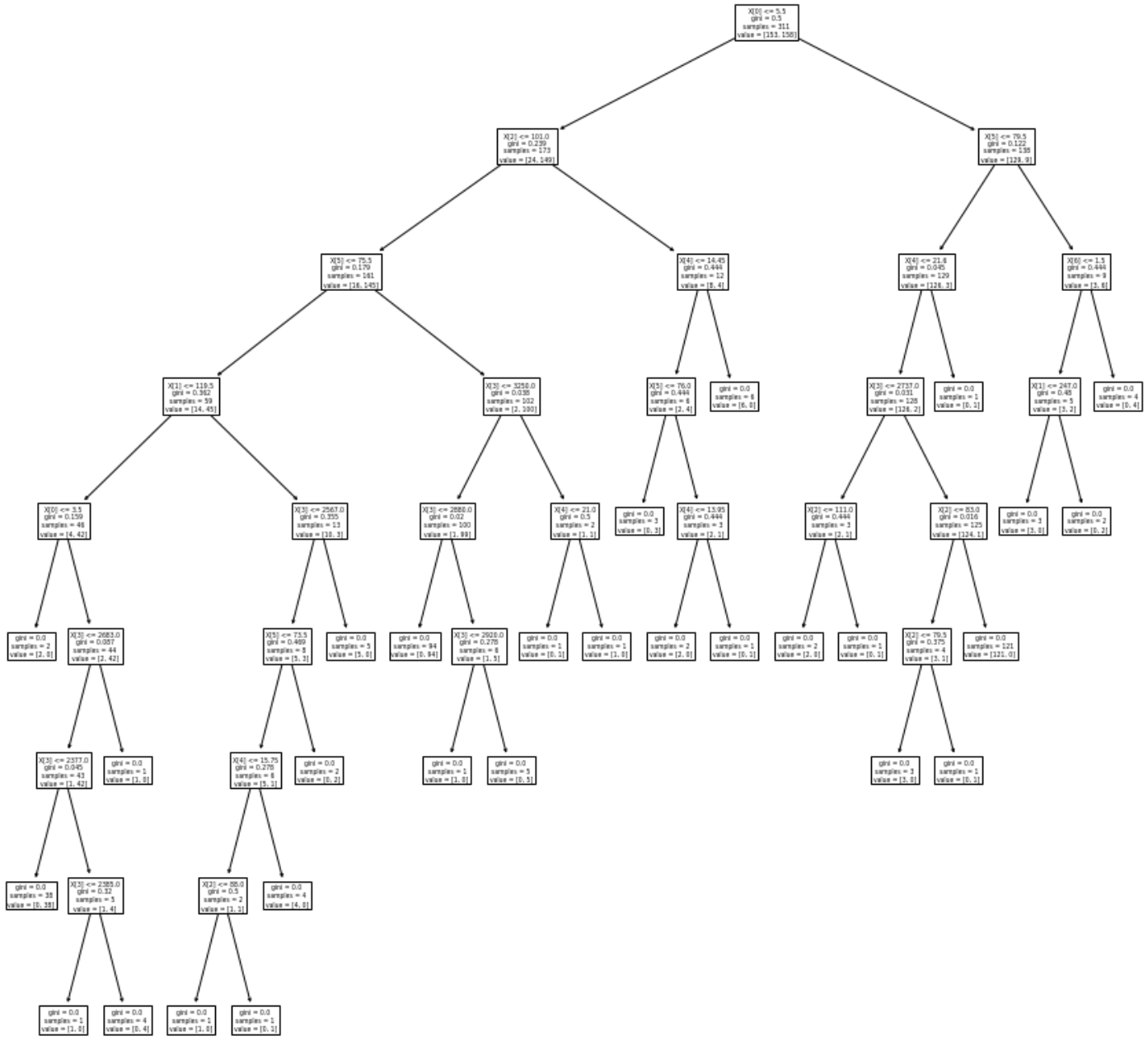
In [113…
```
print(classification_report(Y_test,pred2))
```

```
              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
```

In [113…
```
plt.figure(figsize=(15,15))
tree.plot_tree(clf2)
plt.show()
```



In [113…

```
# 10. Neural Network
```

In [113…

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

In [113…

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

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

In [113…

```
clf3 = MLPClassifier(solver = 'lbfgs', hidden_layer_sizes = (6),max_iter = 1500, random_st
clf3.fit(X_train_scaled,Y_train)
clf3.score(X_train_scaled,Y_train)
```

Out[113…

```
1.0
```

In [113…
```
pred3 = clf3.predict(X_test_scaled)
print(classification_report(Y_test, pred3))
```

```
              precision    recall  f1-score   support

           0       0.96      0.86      0.91        50
           1       0.79      0.93      0.85        28

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

In [113…
```
clf4=MLPClassifier(solver='sgd',hidden_layer_sizes=(5,2),max_iter=1500,random_state=1234)
clf4.fit(X_train_scaled,Y_train)
clf4.score(X_train_scaled,Y_train)
```

Out[113…
```
0.8971061093247589
```

In [114…
```
pred4 = clf4.predict(X_test_scaled)
print(classification_report(Y_test,pred4))
```

```
              precision    recall  f1-score   support

           0       0.93      0.82      0.87        50
           1       0.74      0.89      0.81        28

    accuracy                           0.85        78
   macro avg       0.83      0.86      0.84        78
weighted avg       0.86      0.85      0.85        78
```

In [114…
```
# The first Neural Network performed 3-5% better than the second one.
```

In [ ]:
```
# 11. Analysis
# The best performing algorithm was Logistic Regression, as it has better recall and preci
# For some reason, my Neural Networks were bad. No idea why. Perhaps the data was too simp
# I think I prefer R because it is simpler, and R studio is much easier to navigate than P



-Logistic Regression
    Accuracy = 90
    0 Recall = 84
    1 Recall = 100
    0 Precision = 100
    1 Precision = 78

-Decision Tree
    Accuracy = 90
    0 Recall = 92
    1 Recall = 86
    0 Precision = 92
    1 Precision = 86

-Neural Network 1
    Accuracy = 88
    0 Recall = 86
    1 Recall = 93
    0 Precision = 96
    1 Precision = 79
```

```
-Neural Network 2
    Accuracy = 85
    0 Recall = 82
    1 Recall = 89
    0 Precision = 93
    1 Precision = 74
```