In [182…
```
##IMPORTS##
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.tree import DecisionTreeClassifier, plot_tree
from sklearn import preprocessing
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

In [183…
```
##Part 1
df = pd.read_csv('Auto.csv')
df.head()
```

Out[183]:

	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	forc torinc

In [ ]:
```
```

In [184…
```
print('\nDimensions of the dataframe: ', df.shape)
```

```
Dimensions of the dataframe:  (392, 9)
```

In [185…
```
##Part 2
#MPG Description | Mean: 23.446, Range: 37.6
df['mpg'].describe()
```

Out[185]:
```
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 [186…
```
#Weight Description| Mean: 2977.584, Range: 3527
df['weight'].describe()
```

Out[186]:
```
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 [187…
```
#Year Description | Mean: 76.010, Range: 12
df['year'].describe()
```

Out[187]:
```
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 [188…
```
##Part 3
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 392 entries, 0 to 391
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           392 non-null    float64
 1   cylinders     392 non-null    int64
 2   displacement  392 non-null    float64
 3   horsepower    392 non-null    int64
 4   weight        392 non-null    int64
 5   acceleration  391 non-null    float64
 6   year          390 non-null    float64
 7   origin        392 non-null    int64
 8   name          392 non-null    object
dtypes: float64(4), int64(4), object(1)
memory usage: 27.7+ KB
```

In [189…
```
#Changing cylinders to categorical data
df.cylinders = df.cylinders.astype('category').cat.codes
df.origin = df.origin.astype('category')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 392 entries, 0 to 391
Data columns (total 9 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   mpg           392 non-null    float64
 1   cylinders     392 non-null    int8
 2   displacement  392 non-null    float64
 3   horsepower    392 non-null    int64
 4   weight        392 non-null    int64
 5   acceleration  391 non-null    float64
 6   year          390 non-null    float64
 7   origin        392 non-null    category
 8   name          392 non-null    object
dtypes: category(1), float64(4), int64(2), int8(1), object(1)
memory usage: 22.5+ KB
```

In [190…
```
df.head()
```

Out[190]:

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

In [191…
```
##Part 4
df.isnull().sum()
```

Out[191]:
```
mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    1
year            2
origin          0
name            0
dtype: int64
```

In [192…
```
df = df.dropna()
df.isnull().sum()
```

Out[192]:
```
mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
year            0
origin          0
name            0
dtype: int64
```

In [193…
```
#New dimensions should be around 2-3 smaller than 392
print('\nNew dimensions of the dataframe: ', df.shape)
```

```
New dimensions of the dataframe:  (389, 9)
```

In [194…
```
##Part 5
new_mpg = {"mpg_high": [] }
for item in df.itertuples(index=False):
    new_mpg["mpg_high"].append(1 if item.mpg > df['mpg'].describe()['mean'] else 0)
df.insert(9, "mpg_high", new_mpg["mpg_high"])
```

In [195…
```
df.head()
```

Out[195]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
0	18.0	4	307.0	130	3504	12.0	70.0	1	chevrolet chevelle malibu
1	15.0	4	350.0	165	3693	11.5	70.0	1	buick skylark 320
2	18.0	4	318.0	150	3436	11.0	70.0	1	plymouth satellite
3	16.0	4	304.0	150	3433	12.0	70.0	1	amc rebel sst
6	14.0	4	454.0	220	4354	9.0	70.0	1	chevrolet impala

In [196…
```
df = df.drop(columns=['mpg'])
```

In [197…
```
df.head()
```

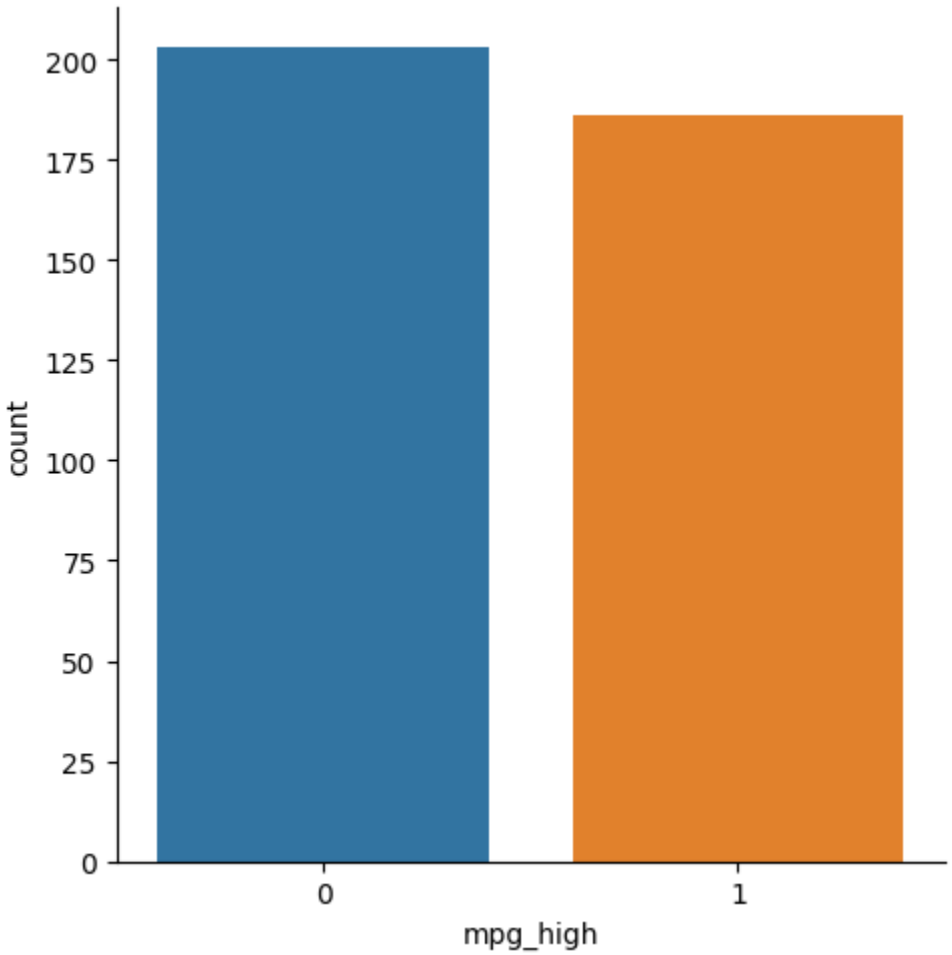
Out[197]:

	cylinders	displacement	horsepower	weight	acceleration	year	origin	name	mpg
0	4	307.0	130	3504	12.0	70.0	1	chevrolet chevelle malibu	
1	4	350.0	165	3693	11.5	70.0	1	buick skylark 320	
2	4	318.0	150	3436	11.0	70.0	1	plymouth satellite	
3	4	304.0	150	3433	12.0	70.0	1	amc rebel sst	
6	4	454.0	220	4354	9.0	70.0	1	chevrolet impala	

In [198…

```
##Part 6
sb.catplot(x="mpg_high", kind='count', data=df)
```

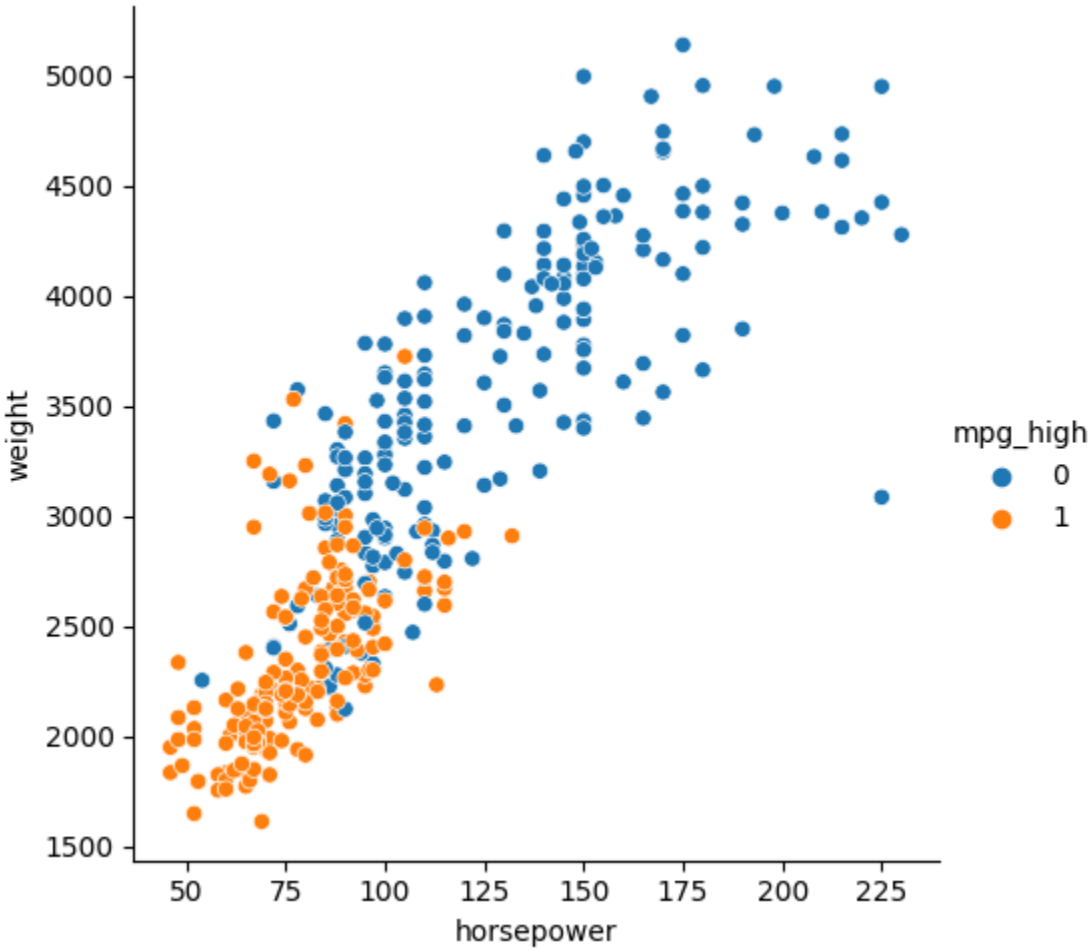
Out[198]: <seaborn.axisgrid.FacetGrid at 0x26b8beb3280>



In [199…

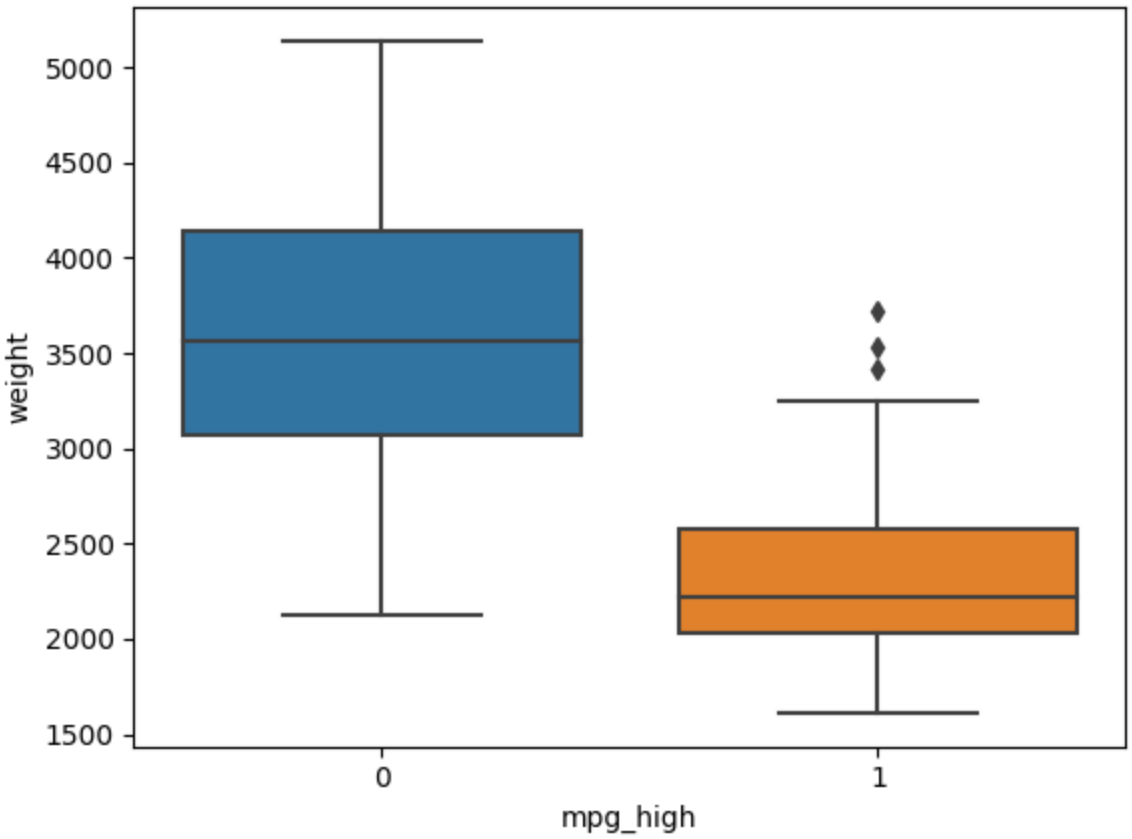
```
#The efficiency of the cars tends to lean less fuel efficient than more fuel effici
sb.relplot(x="horsepower", y="weight", data=df, hue="mpg_high")
```

Out[199]: <seaborn.axisgrid.FacetGrid at 0x26b8d663670>



In [200…
```
#In general, the heavier and higher horsepower the vehicle, the better chance it ha
sb.boxplot(x="mpg_high", y="weight", data=df)
```

Out[200]: <Axes: xlabel='mpg_high', ylabel='weight'>



In [201…
```
#Cars around .5-1 tons will have better fuel efficiency than cars ranging from 1.5-

##Part 7
X = df.iloc[:, 0:7]
y = df.iloc[:, 8]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
print('\nDimensions of X_train: ', X_train.shape)
print('\nDimensions of y_train: ', y_train.shape)
print('\nDimensions of X_test: ', X_test.shape)
print('\nDimensions of Y_test: ', y_test.shape)
```

```
Dimensions of X_train:  (311, 7)

Dimensions of y_train:  (311,)

Dimensions of X_test:  (78, 7)

Dimensions of Y_test:  (78,)
```

In [202…
```
##Part 8
clf = LogisticRegression(solver='lbfgs')
clf.fit(X_train, y_train)
clf.score(X_train, y_train)
```

```
C:\Users\Pickle Mustard\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_q
bz5n2kfra8p0\LocalCache\local-packages\Python39\site-packages\sklearn\linear_model\_
logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[202]: 0.9067524115755627

In [203…
```
pred = clf.predict(X_test)
```

In [204…
```
print('Logistic Regression Accuracy Score: ', accuracy_score(y_test, pred))
print('Logistic Regression Precision Score: ', precision_score(y_test, pred))
print('Logistic Regression Recall Score: ', recall_score(y_test, pred))
print('Logistic Regression F1 Score: ', f1_score(y_test, pred))
```

```
Logistic Regression Accuracy Score:  0.8589743589743589
Logistic Regression Precision Score:  0.7297297297297297
Logistic Regression Recall Score:  0.9642857142857143
Logistic Regression F1 Score:  0.8307692307692307
```

In [205…
```
confusion_matrix(y_test, pred)
```

Out[205]:
```
array([[40, 10],
       [ 1, 27]], dtype=int64)
```

In [206…
```
##Part 9
clf_dt = DecisionTreeClassifier()
clf_dt.fit(X_train, y_train)
pred = clf_dt.predict(X_test)
```

In [207…
```
print('Decision Tree Accuracy Score: ', accuracy_score(y_test, pred))
print('Decision Tree Precision Score: ', precision_score(y_test, pred))
print('Decision Tree Recall Score: ', recall_score(y_test, pred))
print('Decision Tree F1 Score: ', f1_score(y_test, pred))
```

```
Decision Tree Accuracy Score:  0.9230769230769231
Decision Tree Precision Score:  0.8666666666666667
Decision Tree Recall Score:  0.9285714285714286
Decision Tree F1 Score:  0.896551724137931
```

In [208…
```
confusion_matrix(y_test, pred)
```

Out[208]:
```
array([[46,  4],
       [ 2, 26]], dtype=int64)
```

In [209…
```
##Part 10
scaler = preprocessing.StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

clf_nn = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5,2), max_iter=500, rand
clf_nn.fit(X_train_scaled, y_train)
```

Out[209]:
```
MLPClassifier(hidden_layer_sizes=(5, 2), max_iter=500, random_state=1234,
              solver='lbfgs')
```

In [210…
```
pred = clf_nn.predict(X_test_scaled)
print('Accuracy of 5,2 NN: ', accuracy_score(y_test, pred))
print('Precision of 5,2 NN: ', precision_score(y_test, pred))
print('Recall of 5,2 NN: ', recall_score(y_test, pred))
print('F1 of 5,2 NN: ', f1_score(y_test, pred))
confusion_matrix(y_test,pred)
```

```
Accuracy of 5,2 NN:  0.8589743589743589
Precision of 5,2 NN:  0.7575757575757576
Recall of 5,2 NN:  0.8928571428571429
F1 of 5,2 NN:  0.819672131147541
```

Out[210]:
```
array([[42,  8],
       [ 3, 25]], dtype=int64)
```

In [211…
```
clf_nn2 = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(9,9), max_iter=500, ran
clf_nn2.fit(X_train_scaled, y_train)
```

Out[211]:
```
MLPClassifier(hidden_layer_sizes=(9, 9), max_iter=500, random_state=1234,
              solver='lbfgs')
```

In [212…
```
pred = clf_nn2.predict(X_test_scaled)
print('Accuracy of 9,9 NN: ', accuracy_score(y_test, pred))
print('Precision of 9,9 NN: ', precision_score(y_test, pred))
print('Recall of 9,9 NN: ', recall_score(y_test, pred))
print('F1 of 9,9 NN: ', f1_score(y_test, pred))
confusion_matrix(y_test,pred)
```

```
Accuracy of 9,9 NN:  0.8717948717948718
Precision of 9,9 NN:  0.7647058823529411
Recall of 9,9 NN:  0.9285714285714286
F1 of 9,9 NN:  0.8387096774193549
```

Out[212]:
```
array([[42,  8],
       [ 2, 26]], dtype=int64)
```

Topology Comparison

Between the 5,2 Neural network and the 9,9 there was an incredibly slight change in the accuracy of less than .02. The 9,9 topology got 1 additional item correct over the 5,2 topology. Increasing the topology increased the dimensions of the plane that the transformations were being mapped to, making it more accurate to the path the vector of the data would take. It didn't make much difference however as the topology is likely too large for the data set and it not fit correctly for the data size.

Analysis

Between the 3 algorithms tested, the most accurate one was the decision tree with an accuracy around 88-90%, followed by the Neural Network with the larger topology. The logistic regression and smaller neural network trailed at the end, each with an accuracy of around 85%. Precision followed alongside accuracy. The recall score followed an inverse trend of accuracy, with the logistic regression algorithm having the largest score and the 9,9 topology having the smallest.

The likely reason for the Decision-Tree to outperform the other algorithms is that there is an extremely evident trend between a larger weight, horsepower, and the other metrics and the mpg_high classifier to be 0. So a greedy algorithm can easily identify this trend and quickly and accurately pull a heavy car with high horsepower into fuel inefficient and a light car with low horsepower into fuel efficient. Logistic regression might have the dividing line in a place where it can be a bit ambiguous with where cars fall and similarly, the neural networks might have some vehicles that straddle the possibilities of fuel efficiency.

As for my own opinions, I much prefer sklearn and python to R. Maybe its just the familiarity I feel with a programming language more similar to ones that I have years of experience with, but this just felt more natural to type. I think the performance was also better, running in a much better time frame locally than R did much of the time. The documentation is also easier to read. Some of R's documentation felt like it came out of the 90's Web design and was hard to find the information I needed on a function but sklearn had all the parameters what they needed easily reachable.