1. Read the Auto data

```
import pandas as pd
import seaborn as sb
import numpy as np
import tensorflow as tf
from numpy.random import seed
seed(1234)
tf.random.set_seed(1234)
df = pd.read_csv('Auto.csv')
print(df.head())
print('\nData frame dimensions:', 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

   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

Data frame dimensions: (392, 9)
```

2. Data exploration with code

MPG

- Range: 37.6
- Average: 23.445918

```
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
```

Weight

- Range: 37.6
- Average: 2977.584184

```
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
```

Year

- Range: 12
- Average: 76.010256

```
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
```

3. Explore data types

```
df.dtypes
```

```
mpg             float64
cylinders         int64
displacement    float64
horsepower        int64
weight            int64
acceleration    float64
year            float64
origin            int64
```

```
name             object
dtype: object
```

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

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

4. Deal with NAs

```
df.dropna(inplace=True)
df.isnull().sum()
```

```
mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
year            0
origin          0
name            0
dtype: int64
```

```
print('\nNew data frame dimensions:', df.shape)
```

```
New data frame dimensions: (389, 9)
```

5. Modify columns

```
df['mpg_high'] = np.where(df['mpg']> 23.445918, 1, 0)
df.mpg_high = df.mpg_high.astype('category').cat.codes
df.drop(columns=['mpg', 'name'], inplace=True)
print(df.head())
df.dtypes
```

```
   cylinders  displacement  horsepower  weight  acceleration  year origin  \
0          4         307.0         130    3504          12.0  70.0      1
```

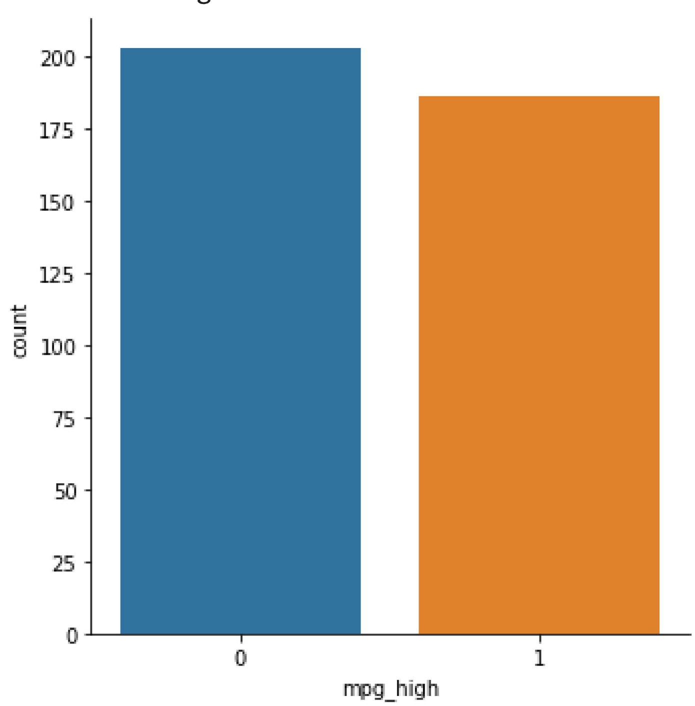
```
1          4        350.0          165     3693           11.5  70.0       1
2          4        318.0          150     3436           11.0  70.0       1
3          4        304.0          150     3433           12.0  70.0       1
6          4        454.0          220     4354            9.0  70.0       1

   mpg_high
0         0
1         0
2         0
3         0
6         0
cylinders          int8
displacement    float64
horsepower        int64
weight            int64
acceleration    float64
year            float64
origin         category
mpg_high           int8
dtype: object
```

6. Data exploration with graphs

The number above the average mp is similar to the number below which was expected when splitting on the mean. Less above suggests that the mean is skewed a bit higher.

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

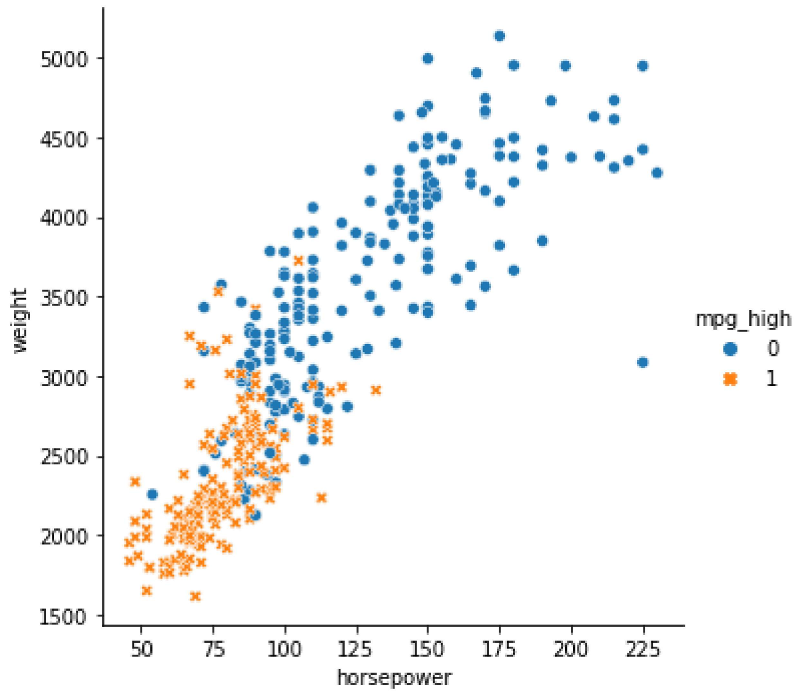
```
<seaborn.axisgrid.FacetGrid at 0x7fbfe294f450>
```



Car weight, horsepower, and mpg are all directly related.

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

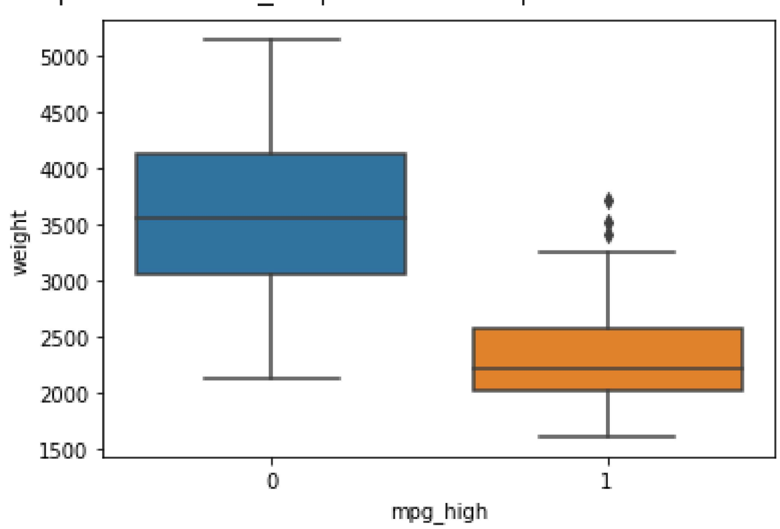
```
<seaborn.axisgrid.FacetGrid at 0x7fbfe004ef90>
```



More efficient cars tend to be lighter than less efficient cars. The cars which have a higher than average mpg had a smaller range and standard deviation, but had a few outliers.

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbfdfb5b690>
```



7. Train/test split

```
from sklearn.model_selection import train_test_split
X = df.iloc[:,0:6]
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, 6)
test size: (78, 6)
```

8. Logistic Regression

```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression(solver='lbfgs')
lr.fit(X_train, y_train)
lr.score(X_train, y_train)
```

```
0.9035369774919614
```

```
from sklearn.metrics import classification_report
pred1 = lr.predict(X_test)
print(classification_report(y_test, pred1))
```

```
              precision    recall  f1-score   support

           0       0.98      0.80      0.88        50
           1       0.73      0.96      0.83        28

    accuracy                           0.86        78
   macro avg       0.85      0.88      0.85        78
weighted avg       0.89      0.86      0.86        78
```

9. Decision Tree

```
from sklearn.tree import DecisionTreeClassifier

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

```
DecisionTreeClassifier()
```

```
pred2 = dt.predict(X_test)
```

```
print(classification_report(y_test, pred2))
```

```
              precision    recall  f1-score   support

           0       0.96      0.90      0.93        50
           1       0.84      0.93      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
```

```
import graphviz
from sklearn import tree
dot_data = tree.export_graphviz(dt, out_file=None, feature_names=X.columns,  filled=True, rou
graph = graphviz.Source(dot_data)
graph
```

10. Neural Network

```
from sklearn import preprocessing

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

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

Multi-layer Perceptron classifier

```
from sklearn.neural_network import MLPClassifier

clf = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5, 2), max_iter=500, random_state=12
clf.fit(X_train_scaled, y_train)
```

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

```
pred3 = clf.predict(X_test_scaled)
print(classification_report(y_test, pred3))
```

```
              precision    recall  f1-score   support

           0       0.94      0.88      0.91        50
           1       0.81      0.89      0.85        28

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

```
weighted avg       0.89      0.88      0.89        78
```

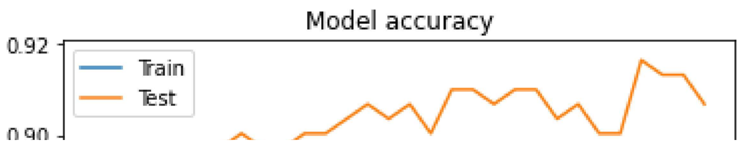
Keras

```
from __future__ import print_function
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import RMSprop

y_train = keras.utils.to_categorical(y_train, 2)
y_test2 = keras.utils.to_categorical(y_test, 2)

batch_size = 128
epochs = 30

model = Sequential()
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(2, activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer=RMSprop(),metrics=['accuracy'])

history = model.fit(X_train_scaled, y_train,batch_size=batch_size,epochs=epochs,verbose=1,val
```

```
Epoch 1/30
3/3 [==============================] - 1s 102ms/step - loss: 0.6078 - accuracy: 0.82
Epoch 2/30
3/3 [==============================] - 0s 18ms/step - loss: 0.4905 - accuracy: 0.893
Epoch 3/30
3/3 [==============================] - 0s 18ms/step - loss: 0.4296 - accuracy: 0.897
Epoch 4/30
3/3 [==============================] - 0s 26ms/step - loss: 0.3857 - accuracy: 0.893
Epoch 5/30
3/3 [==============================] - 0s 20ms/step - loss: 0.3545 - accuracy: 0.890
Epoch 6/30
3/3 [==============================] - 0s 17ms/step - loss: 0.3302 - accuracy: 0.893
Epoch 7/30
3/3 [==============================] - 0s 16ms/step - loss: 0.3111 - accuracy: 0.897
Epoch 8/30
3/3 [==============================] - 0s 17ms/step - loss: 0.2949 - accuracy: 0.900
Epoch 9/30
3/3 [==============================] - 0s 17ms/step - loss: 0.2843 - accuracy: 0.897
Epoch 10/30
3/3 [==============================] - 0s 17ms/step - loss: 0.2745 - accuracy: 0.897
Epoch 11/30
3/3 [==============================] - 0s 16ms/step - loss: 0.2635 - accuracy: 0.900
Epoch 12/30
3/3 [==============================] - 0s 20ms/step - loss: 0.2578 - accuracy: 0.900
Epoch 13/30
3/3 [==============================] - 0s 17ms/step - loss: 0.2538 - accuracy: 0.903
```

```
Epoch 14/30
3/3 [==============================] - 0s 18ms/step - loss: 0.2473 - accuracy: 0.906
Epoch 15/30
3/3 [==============================] - 0s 19ms/step - loss: 0.2395 - accuracy: 0.903
Epoch 16/30
3/3 [==============================] - 0s 17ms/step - loss: 0.2359 - accuracy: 0.906
Epoch 17/30
3/3 [==============================] - 0s 16ms/step - loss: 0.2300 - accuracy: 0.900
Epoch 18/30
3/3 [==============================] - 0s 16ms/step - loss: 0.2257 - accuracy: 0.910
Epoch 19/30
3/3 [==============================] - 0s 17ms/step - loss: 0.2259 - accuracy: 0.910
Epoch 20/30
3/3 [==============================] - 0s 16ms/step - loss: 0.2191 - accuracy: 0.906
Epoch 21/30
3/3 [==============================] - 0s 20ms/step - loss: 0.2197 - accuracy: 0.910
Epoch 22/30
3/3 [==============================] - 0s 26ms/step - loss: 0.2122 - accuracy: 0.910
Epoch 23/30
3/3 [==============================] - 0s 15ms/step - loss: 0.2134 - accuracy: 0.903
Epoch 24/30
3/3 [==============================] - 0s 18ms/step - loss: 0.2081 - accuracy: 0.906
Epoch 25/30
3/3 [==============================] - 0s 19ms/step - loss: 0.2085 - accuracy: 0.900
Epoch 26/30
3/3 [==============================] - 0s 18ms/step - loss: 0.2034 - accuracy: 0.900
Epoch 27/30
3/3 [==============================] - 0s 14ms/step - loss: 0.1990 - accuracy: 0.916
Epoch 28/30
3/3 [==============================] - 0s 16ms/step - loss: 0.2006 - accuracy: 0.913
```

```
import matplotlib.pyplot as plt

plt.plot(history.history['val_accuracy'])
plt.plot(history.history['accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```




```
score = model.evaluate(X_test_scaled, y_test2, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.21548594534397125
Test accuracy: 0.8846153616905212
```


```
pred4 = np.argmax(model.predict(X_test_scaled),axis=1)
print(classification_report(y_test, pred4))
```

```
3/3 [==============================] - 0s 5ms/step
              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
```

Comparison

```
print(classification_report(y_test, pred3))
print(classification_report(y_test, pred4))
```

```
              precision    recall  f1-score   support

           0       0.94      0.88      0.91        50
           1       0.81      0.89      0.85        28

    accuracy                           0.88        78
   macro avg       0.87      0.89      0.88        78
weighted avg       0.89      0.88      0.89        78
              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 final accuracy was the same for both, however the other supporting stats were different. I increased MLP epochs to 30 which felt fair and ended up with the same accuracy for both neural networks. There are so many knobs that you can turn for neural networks that its difficult to tell when you have the best solution.

11. Analysis

```
print(classification_report(y_test, pred1))
print(classification_report(y_test, pred2))
print(classification_report(y_test, pred3))
print(classification_report(y_test, pred4))
```

```
              precision    recall  f1-score   support

           0       0.98      0.80      0.88        50
           1       0.73      0.96      0.83        28

    accuracy                           0.86        78
   macro avg       0.85      0.88      0.85        78
weighted avg       0.89      0.86      0.86        78

              precision    recall  f1-score   support

           0       0.96      0.90      0.93        50
           1       0.84      0.93      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

              precision    recall  f1-score   support

           0       0.94      0.88      0.91        50
           1       0.81      0.89      0.85        28

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

              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 best performing algorithm in terms of accuracy was the decision tree. The algorithms had higher precision for finding below average mpg cars than above, while the inverse was true for recall. I think that decision tree outperformed the others because the data wasn't quite linear which made logistic regression worse and my inexperience with neural networks. The neural networks were close behind and if I had more experience they probably would have ended up being the best which is one downside of using more complex algorithms.

SKLearn through Google Colab is online and hosted through Google's hardware which makes switching computers easy and has good performance at the cost of always needing an internet connection. I always felt forced to use my laptop even when I was home when using R because it is annoying to switch. Having it online is also great for working in a group but I didn't work in a group for any of my assignments.