```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mour
```

First we import the data and run df.head() to check the first couple of entries.

```
import pandas as pd
df = pd.read_csv('/content/drive/MyDrive/ML/Auto.csv')
df.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	nam
0	18.0	8	307.0	130	3504	12.0	70.0	1	chevrol chevel malit
1	15.0	8	350.0	165	3693	11.5	70.0	1	buid skyla 32
									plymou

Then we show the dimensions of the data -

```
df.shape
```

```
(392, 9)
```

Now for data exploration using describe()

```
df_1 = df[['mpg','weight','year']]
df_1.describe()
#        MPG     CYLINDERS DISPLACEMENT HORSEPOWER WEIGHT    ACCELERATION YEAR ORIGIN
#RANGE   37      5         387          184        3527      16           12   2
#AVG     23.446  5.47      194.412      104.469    2977.584  15.554       76   1.576
```

	mpg	weight	year
count	392.000000	392.000000	390.000000
mean	23.445918	2977.584184	76.010256
std	7.805007	849.402560	3.668093

Then we explore the data types and make a few adjustments

```
df.dtypes
```

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

```
df_copy1 = df.copy()
df_copy1.cylinders = df.cylinders.astype('category').cat.codes
df_copy1.origin = df.origin.astype('category')

print(df_copy1.dtypes, "\n")
df_copy1.head()

df = df_copy1
```

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

Now we go ahead and clean up the NAs

```
df = df.dropna()
print("Dimensions post NA drop: ", df.shape)
```

```
Dimensions post NA drop:  (389, 9)
```

Let's go ahead and add a new column, mpg_high where it checks if mpg > avg

```
mpg_avg = df['mpg'].mean()

def mpg_vs_avg(row):
  if row['mpg'] > mpg_avg:
    return 1
  return 0

df_copy2 = df.copy()
df_copy2['mpg_high'] = df_copy2.apply(lambda row: mpg_vs_avg(row), axis = 1)
df_copy2.mpg_high = df_copy2.mpg_high.astype('category')

df_copy2 = df_copy2[['cylinders','displacement','horsepower','weight','acceleration','year','

df_copy2.head()
```

	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

Now we're going to do some data exploration with graphs using seaborn

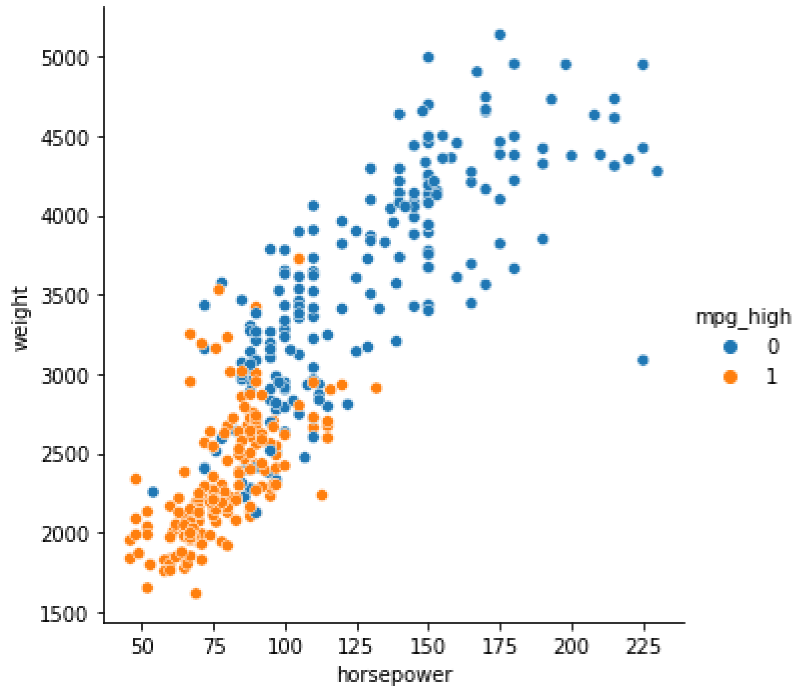
```
import seaborn as sb
sb.catplot(x='mpg_high', kind='count', data=df_copy2)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f31447757d0>
```

Looks like most of the autos are below the average mpg

```
sb.relplot(x='horsepower', y='weight', data=df_copy2, hue='mpg_high')
```

```
<seaborn.axisgrid.FacetGrid at 0x7f3144958d90>
```



In general, lower horsepower and lower weight looks to trend the mpg above the average

```
sb.boxplot('mpg_high', y='weight', data=df_copy2)
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass tl
  FutureWarning
<matplotlib.axes._subplots.AxesSubplot at 0x7f3144684490>
```

There are mostly outliers lying where the mpg is higher than the average in terms of weight - in general, lower weight implies higher than average mpg.

Let's go ahead and start splitting train / test

```
X = df_copy2[['cylinders','displacement','horsepower','weight','acceleration','year','origin'
y = df_copy2['mpg_high']
from sklearn.model_selection import train_test_split
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, 7)
test size: (78, 7)
```

Logistic Regression

Now we do Logistic Regression

```
from sklearn.linear_model import LogisticRegression

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

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Convergenc
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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
0.9067524115755627
```

```
# predictions

pred = clf.predict(X_test)
```

Let's check the metrics of the prediction

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

```
print('accuracy score: ', accuracy_score(y_test, pred))

# confusion matrix
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, pred)
```

```
accuracy score:  0.8589743589743589
array([[40, 10],
       [ 1, 27]])
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))
```

```
              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
```

Decision Tree

Let's go ahead and move onto the decision trees

```
from sklearn.tree import DecisionTreeClassifier

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

```
DecisionTreeClassifier()
```

```
# predictions

pred = clf.predict(X_test)
```

These are the metrics for the decision tree

```
# evaluation
print('accuracy score: ', accuracy_score(y_test, pred))

# confusion matrix
confusion_matrix(y_test, pred)
```

```
accuracy score:  0.8974358974358975
array([[46,  4],
       [ 4, 24]])
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.92      0.92      0.92        50
           1       0.86      0.86      0.86        28

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

Neural Network

Finally, let's try a neural network - first we need to normalize the data

```
# normalization
from sklearn import preprocessing

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

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

```
from sklearn.neural_network import MLPClassifier

clf = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5, 2), max_iter=500, random_state=123
clf.fit(X_train_scaled, y_train)
```

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

```
# predictions

pred = clf.predict(X_test_scaled)
```

Then we check the accuracy and the confusion matrix

```
print('accuracy = ', accuracy_score(y_test,pred))
confusion_matrix(y_test, pred)
```

```
accuracy =  0.8717948717948718
array([[43,  7],
       [ 3, 25]])
```

```
from sklearn.metrics import classification_report
print(classification_report(y_test, 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
```

Let's try another variation on this neural network

```
clf = MLPClassifier(solver='sgd', hidden_layer_sizes=(7,), max_iter=750, random_state=1234)
clf.fit(X_train_scaled, y_train)
```

```
MLPClassifier(hidden_layer_sizes=(7,), max_iter=750, random_state=1234,
              solver='sgd')
```

```
# predictions

pred = clf.predict(X_test_scaled)

print('accuracy = ', accuracy_score(y_test,pred))
confusion_matrix(y_test, pred)
```

```
accuracy =  0.8461538461538461
array([[41,  9],
       [ 3, 25]])
```

```
print(classification_report(y_test, pred))
```

```
              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 terms of neural networks, the first one performed better than the second one most likely due to the difference in hidden_layer_sizes, since that is the main point of difference between the two. The topology itself doesn't change too much of the results

Analysis

- The algorithm that performed the best was actually the Decision tree
- In terms of accuracy and recall score, DT > NN1 > LOG > NN2
- In terms of precision score, DT > LOG > NN1 > NN2

Decision Tree probably performed the best because the algorithm for decision trees is not too complicated and as such doesn't overfit too much nor try to account for too much in the data.

To be completely honest, R versus sklearn was not all that different, since I myself don't have that much Python experience. I will say that importing packages was way mroe convenient and quick in Python, and the documentation with these Python libraries was also very nice. The syntax of both were smilar enough to not make too much of a difference. I also liked using Google Colab more than R Studio, if not for the sole purpose that it's online-based; the only gripe I had with using Google Colab was that it was a little bit more finnicky to import files in but overall not that bad.