```
import pandas as pd
import matplotlib.pyplot as plt

# read in the data
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from sklearn.tree import DecisionTreeClassifier
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
import numpy as np
import tensorflow as tf

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

# output the first few rows
print(df.head())

# output the dimensions of the data
print(df.shape)

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

#The Auto dataset contains information on the miles per gallon, weight, and year of manufacture for 392 different cars.
# The data shows that the range of miles per gallon is quite wide, with some cars getting as little as 9 miles per
# gallon and others getting as much as 46.6 miles per gallon. The average miles per gallon for the dataset is 23.45,
# with a standard deviation of 7.81, indicating that there is quite a bit of variability in fuel efficiency across the
# cars in the dataset. The weight of the cars ranges from 1613 to 5140 pounds,
# with an average weight of 2977.58 pounds. Finally, the year of manufacture ranges from 70 to 82, with an average of 76.01.

# Check the data types of all columns
print(df.dtypes)

# Change the cylinders column to categorical using cat.codes
df['cylinders'] = df['cylinders'].astype('category').cat.codes

# Change the origin column to categorical without using cat.codes
df['origin'] = df['origin'].astype('category')

# Verify the changes with the dtypes attribute
print(df.dtypes)

# Drop rows with NAs
df.dropna(inplace=True)
```

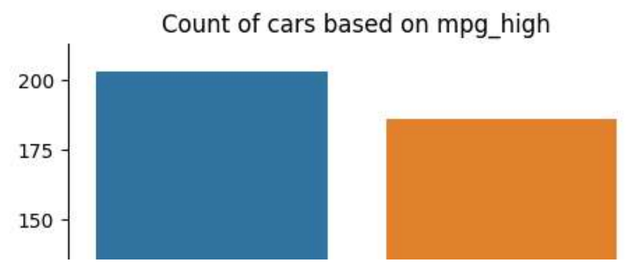
```
    mpg  cylinders  displacement  horsepower  weight  acceleration  year  \
0  18.0          8         307.0         130    3504          12.0  70.0
1  15.0          8         350.0         165    3693          11.5  70.0
2  18.0          8         318.0         150    3436          11.0  70.0
3  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
(392, 9)
              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
     mpg             float64
     cylinders         int64
     displacement    float64
     horsepower        int64
     weight            int64
     acceleration    float64
     year            float64
     origin            int64
     name             object
     dtype: object
     mpg              float64
     cylinders           int8
     displacement     float64
     horsepower         int64
     weight             int64
     acceleration     float64
     year             float64
     origin          category
     name              object
     dtype: object
```

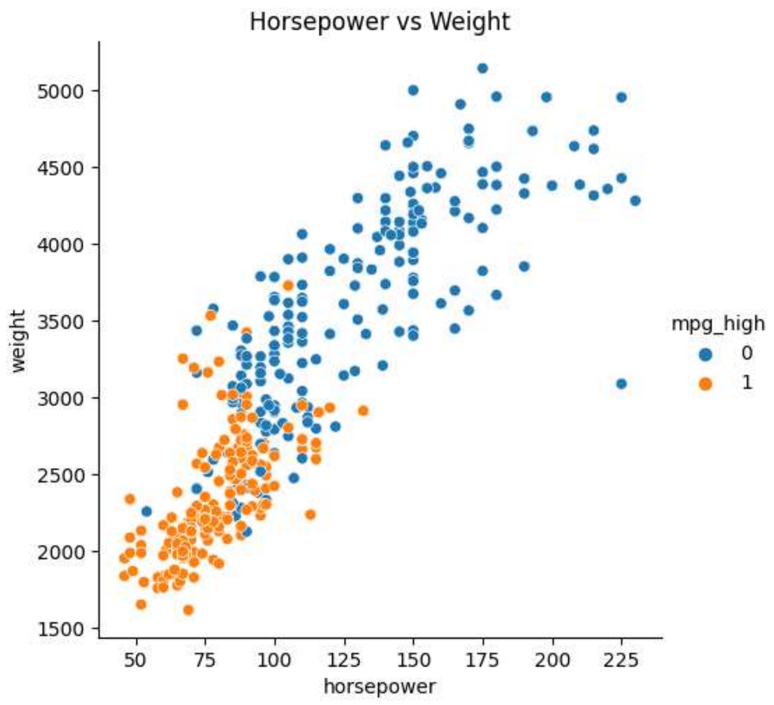
```
# Output the new dimensions
print(df.shape)

# create mpg_high column
avg_mpg = df['mpg'].mean()
df['mpg_high'] = (df['mpg'] > avg_mpg).astype(int).astype('category')

# drop mpg and name columns
df.drop(columns=['mpg', 'name'], inplace=True)

# output first few rows
print(df.head())
```

```
     (389, 9)
        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
```

```
#First graph
sns.catplot(x="mpg_high", kind="count", data=df)
plt.title("Count of cars based on mpg_high")
plt.show()
#I learned that the 0s and1s are close to equal but there are more zeroes.
```



```
#Second graph
sns.relplot(x="horsepower", y="weight", hue="mpg_high", data=df)
plt.title("Horsepower vs Weight")
plt.show()
#From this graph I learned that the weight of the zeroes is on average much higher than the weight of the ones. So
#the cars with lower mpg have higher horsepower and weight.
```



Double-click (or enter) to edit

Double-click (or enter) to edit

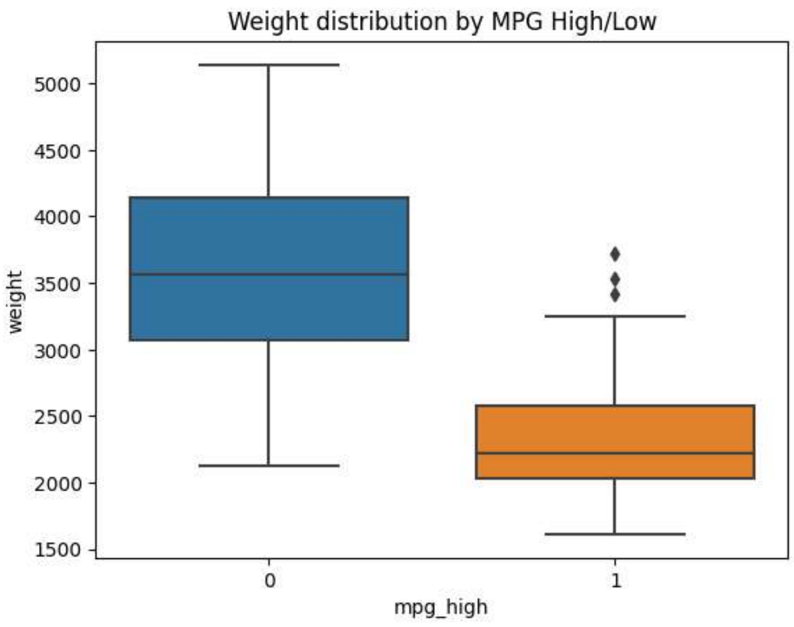
Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

```
#Third graph
sns.boxplot(x="mpg_high", y="weight", data=df)
plt.title("Weight distribution by MPG High/Low")
plt.show()
#The cars with high mpg have lower weight
```



```
# set random seed
seed = 1234

# split data into X and y
X = df.drop('mpg_high', axis=1)
Y = df['mpg_high']

# split data into train and test sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=seed)

# output dimensions of train and test sets
print("Train dimensions:", X_train.shape)
print("Test dimensions:", X_test.shape)
```

```
Train dimensions: (311, 7)
Test dimensions: (78, 7)
```

```
# Train the logistic regression model
lr = LogisticRegression(solver='lbfgs', random_state=seed)
lr.fit(X_train, Y_train)

# Test the logistic regression model and evaluate its performance
y_pred = lr.predict(X_test)
accuracy = accuracy_score(Y_test, y_pred)
print('Accuracy:', accuracy)

report = classification_report(Y_test, y_pred)
print('Classification Report:\n', report)
```

```
Accuracy: 0.8589743589743589
Classification Report:
               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

/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (statu
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(
```

```
# train decision tree
dt = DecisionTreeClassifier(random_state=seed)
dt.fit(X_train, Y_train)

# predict on test set
y_pred = dt.predict(X_test)

# evaluate model performance
print(classification_report(Y_test, y_pred))
```

```
              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
```

```
model1 = Sequential()
model1.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
model1.add(Dense(64, activation='relu'))
model1.add(Dense(32, activation='relu'))
model1.add(Dense(32, activation='relu'))
model1.add(Dense(1, activation='sigmoid'))

# Compile the model
model1.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.001), metrics=['accuracy'])

# Train the model
model12 = model1.fit(X_train, Y_train, epochs=50, validation_data=(X_test, Y_test))

# predict the classes of the test set using model1
y_pred = model1.predict(X_test)

y_pred1_classes = np.round(y_pred)

# print the classification report
print(classification_report(Y_test, y_pred1_classes))
```

```
10/10 [==============================] - 0s 8ms/step - loss: 0.5628 - accuracy: 0.7781 - val_loss: 0.4894 - val_accuracy: 0.84
Epoch 42/50
10/10 [==============================] - 0s 8ms/step - loss: 1.0846 - accuracy: 0.6720 - val_loss: 0.3465 - val_accuracy: 0.88
Epoch 43/50
10/10 [==============================] - 0s 5ms/step - loss: 0.6896 - accuracy: 0.7814 - val_loss: 0.3041 - val_accuracy: 0.84
Epoch 44/50
10/10 [==============================] - 0s 6ms/step - loss: 0.3193 - accuracy: 0.8875 - val_loss: 0.3281 - val_accuracy: 0.85
Epoch 45/50
10/10 [==============================] - 0s 7ms/step - loss: 0.4214 - accuracy: 0.8650 - val_loss: 0.5785 - val_accuracy: 0.82
Epoch 46/50
10/10 [==============================] - 0s 8ms/step - loss: 0.6463 - accuracy: 0.7878 - val_loss: 0.6497 - val_accuracy: 0.79
Epoch 47/50
10/10 [==============================] - 0s 7ms/step - loss: 0.6112 - accuracy: 0.7974 - val_loss: 0.4314 - val_accuracy: 0.87
Epoch 48/50
10/10 [==============================] - 0s 8ms/step - loss: 0.3101 - accuracy: 0.8875 - val_loss: 0.2777 - val_accuracy: 0.85
Epoch 49/50
10/10 [==============================] - 0s 6ms/step - loss: 0.3394 - accuracy: 0.8553 - val_loss: 0.2725 - val_accuracy: 0.85
Epoch 50/50
10/10 [==============================] - 0s 6ms/step - loss: 0.4039 - accuracy: 0.8392 - val_loss: 0.9199 - val_accuracy: 0.70
3/3 [==============================] - 0s 2ms/step
              precision    recall  f1-score   support

           0       1.00      0.54      0.70        50
           1       0.55      1.00      0.71        28

    accuracy                           0.71        78
   macro avg       0.77      0.77      0.71        78
weighted avg       0.84      0.71      0.70        78
```

```
from keras.optimizers import SGD

# Define the second model with a different topology
model2 = Sequential()
model2.add(Dense(8, input_dim=X_train.shape[1], activation='relu'))
model2.add(Dense(16, activation='relu'))
model2.add(Dense(16, activation='relu'))
model2.add(Dense(16, activation='relu'))
model2.add(Dense(1, activation='sigmoid'))

sgd = SGD(learning_rate=0.01, momentum=0.9)
model2.compile(loss='binary_crossentropy', optimizer=sgd, metrics=['accuracy'])

model2.fit(X_train, Y_train, epochs=50, validation_data=(X_test, Y_test))

# predict the classes of the test set using model1
y_pred = model2.predict(X_test)

y_pred2_classes = np.round(y_pred)

# print the classification report
print(classification_report(Y_test, y_pred2_classes,zero_division=1))
```

```
Epoch 40/50
10/10 [==============================] - 0s 4ms/step - loss: 0.6931 - accuracy: 0.5080 - val_loss: 0.6987 - val_accuracy: 0.35
Epoch 41/50
10/10 [==============================] - 0s 4ms/step - loss: 0.6932 - accuracy: 0.5080 - val_loss: 0.6988 - val_accuracy: 0.35
Epoch 42/50
10/10 [==============================] - 0s 4ms/step - loss: 0.6932 - accuracy: 0.5080 - val_loss: 0.6991 - val_accuracy: 0.35
Epoch 43/50
10/10 [==============================] - 0s 5ms/step - loss: 0.6932 - accuracy: 0.5080 - val_loss: 0.7005 - val_accuracy: 0.35
Epoch 44/50
10/10 [==============================] - 0s 4ms/step - loss: 0.6931 - accuracy: 0.5080 - val_loss: 0.6986 - val_accuracy: 0.35
Epoch 45/50
10/10 [==============================] - 0s 4ms/step - loss: 0.6931 - accuracy: 0.5080 - val_loss: 0.6978 - val_accuracy: 0.35
Epoch 46/50
10/10 [==============================] - 0s 4ms/step - loss: 0.6932 - accuracy: 0.5080 - val_loss: 0.6993 - val_accuracy: 0.35
Epoch 47/50
10/10 [==============================] - 0s 6ms/step - loss: 0.6932 - accuracy: 0.5080 - val_loss: 0.6977 - val_accuracy: 0.35
Epoch 48/50
10/10 [==============================] - 0s 6ms/step - loss: 0.6930 - accuracy: 0.5080 - val_loss: 0.6973 - val_accuracy: 0.35
Epoch 49/50
10/10 [==============================] - 0s 6ms/step - loss: 0.6931 - accuracy: 0.5080 - val_loss: 0.6980 - val_accuracy: 0.35
Epoch 50/50
10/10 [==============================] - 0s 5ms/step - loss: 0.6931 - accuracy: 0.5080 - val_loss: 0.6969 - val_accuracy: 0.35
3/3 [==============================] - 0s 4ms/step
              precision    recall  f1-score   support

           0       1.00      0.00      0.00        50
           1       0.36      1.00      0.53        28

    accuracy                           0.36        78
   macro avg       0.68      0.50      0.26        78
weighted avg       0.77      0.36      0.19        78
```

Analysis

The decision tree performed the best followed by the logisitic regression and then the neural network.

This could be because the decision tree has the ability to easily capture non-linear relationships. I also think that I did not use the right neural network setttings because of the very low accuracy. The data might have had features that were uable to be captured linearly.

The logistic regression had an accuracy of 86%, and the precision and recall for class 0 are 0.98 and 0.80, respectively, while for class 1, the precision and recall are 0.73 and 0.96, respectively.

The decision tree, for class 0, the precision is 0.96, recall is 0.92, and f1-score is 0.94.

For class 1, the precision is 0.87, recall is 0.93, and f1-score is 0.90.

The overall accuracy is 0.92.

For the neural netowrks, the first network has a higher recall for class 1 (1.00 vs 0.93) and a higher f1-score for class 1 (0.53 vs 0.71) but it has a much lower accuracy (0.36 vs 0.71) compared to the second network. Additionally, the first network has a much lower precision for class 0 (1.00 vs 0.36) and an f1-score of 0.00 for class 0, indicating that it is performing poorly in detecting class 0. On the other hand, the second network has a better precision for class 1 (0.55 vs 0.36) and a higher f1-score for class 0 (0.70 vs 0.19), indicating that it is better at detecting both classes overall.

I think the decsion tree was able to complete the more complex linear relationships beteer than the other models and I was not able to find the right network settings to get a better accuracy.

In my experience, using sklearn was more straightforward and easier

to use compared to R. The sklearn library has a more consistent API and

is better documented than R's machine learning libraries. The sklearn library also has a wider range of

algorithms and tools available

for machine learning tasks.