In [ ]:
```
# Import necessary libraries
import pandas as pd
import seaborn as sns



# 1. Read the Auto data
# a. use pandas to read the data
auto_data = pd.read_csv('Auto.csv')

# b. output the first few rows
print(auto_data.head())

# c. output the dimensions of the data
print(auto_data.shape)

# 2. Data exploration with code
# a. use describe() on the mpg, weight, and year columns
cols_to_describe = ['mpg', 'weight', 'year']
desc = auto_data[cols_to_describe].describe()
print(desc)

# b. write comments indicating the range and average of each column
# Ranges: mpg (min-max), weight (min-max), year (min-max)
# Averages: mpg (mean), weight (mean), year (mean)

# 3. Explore data types
# a. check the data types of all columns
print(auto_data.dtypes)

# b. change the cylinders column to categorical (use cat.codes)
auto_data['cylinders'] = auto_data['cylinders'].astype('category').cat.codes

# c. change the origin column to categorical (don't use cat.codes)
auto_data['origin'] = auto_data['origin'].astype('category')

# d. verify the changes with the dtypes attribute
print(auto_data.dtypes)

# 4. Deal with NAs
# a. delete rows with NAs
auto_data = auto_data.dropna()

# b. output the new dimensions
print(auto_data.shape)

# 5. Modify columns
# a. make a new column, mpg_high, and make it categorical
mpg_mean = auto_data['mpg'].mean()
auto_data['mpg_high'] = (auto_data['mpg'] > mpg_mean).astype(int)

# b. delete the mpg and name columns
auto_data = auto_data.drop(['mpg', 'name'], axis=1)

# c. output the first few rows of the modified data frame
print(auto_data.head())
```

```
# 6. Data exploration with graphs
# a. seaborn catplot on the mpg_high column
sns.catplot(x='mpg_high', data=auto_data)

# b. seaborn relplot with horsepower on the x axis, weight on the y axis, setting hue or
sns.relplot(x='horsepower', y='weight', hue='mpg_high', data=auto_data)

# c. seaborn boxplot with mpg_high on the x axis and weight on the y axis
sns.boxplot(x='mpg_high', y='weight', data=auto_data)

# d. for each graph, write a comment indicating one thing you learned about the data fro
# Graph 1: The distribution of cars in low and high mpg categories
# Graph 2: Cars with higher mpg tend to have lower horsepower and weight
# Graph 3: Cars with high mpg have a lower median weight and smaller range compared to t

# 7. Train/test split (5 points)
from sklearn.model_selection import train_test_split

X = auto_data.drop(columns=['mpg_high'])
y = auto_data['mpg_high']

# a. 80/20
# b. use seed 1234 so we all get the same results
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=12

# c. train /test X data frames consists of all remaining columns except mpg_high
# d. output the dimensions of train and test
print("Train dimensions:", X_train.shape)
print("Test dimensions:", X_test.shape)

# 8. Logistic Regression (10 points)
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

# a. train a logistic regression model using solver lbfgs
logreg = LogisticRegression(solver='lbfgs')
logreg.fit(X_train, y_train)

# b. test and evaluate
y_pred_logreg = logreg.predict(X_test)

# c. print metrics using the classification report
print("Logistic Regression classification report:")
print(classification_report(y_test, y_pred_logreg, zero_division=0))

# 9. Decision Tree (10 points)
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt

# a. train a decision tree
tree = DecisionTreeClassifier()
tree.fit(X_train, y_train)

# b. test and evaluate
y_pred_tree = tree.predict(X_test)

# c. print the classification report metrics
print("Decision Tree classification report:")
print(classification_report(y_test, y_pred_tree, zero_division=0))
```

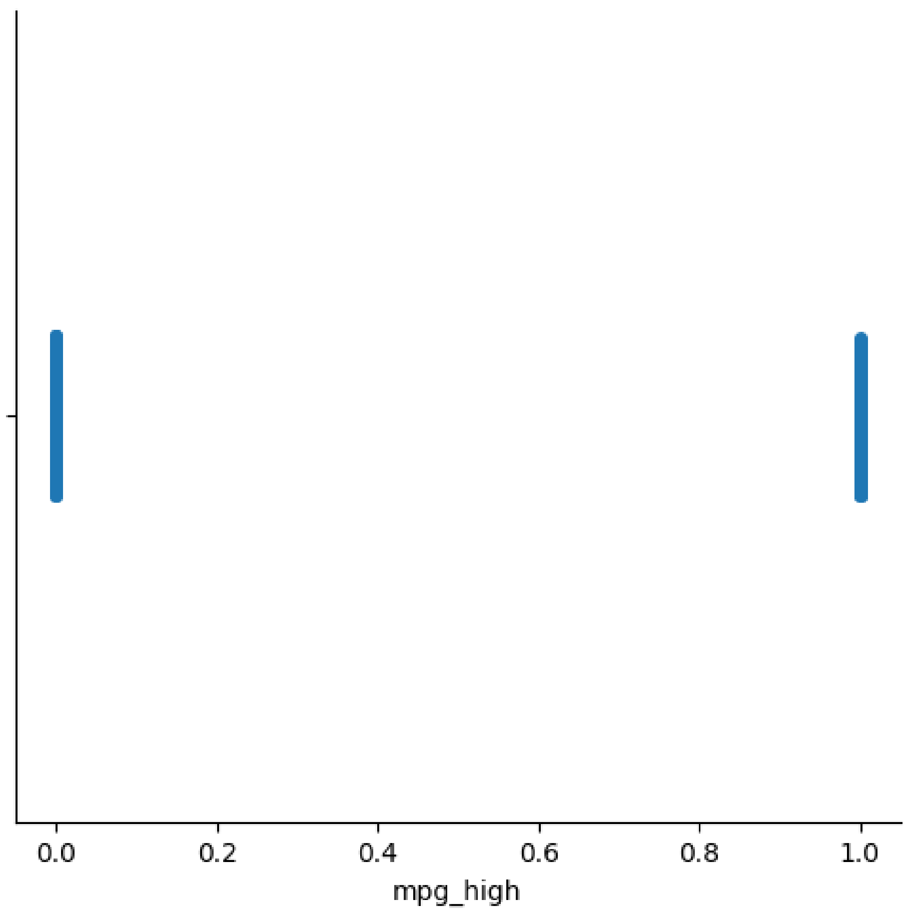
```
# d. plot the tree (optional, see: https://scikit-learn.org/stable/modules/tree.html)
plt.figure(figsize=(12, 8))
plot_tree(tree, filled=True, feature_names=X.columns, class_names=["Low", "High"])
plt.show()

# 10. Neural Network (15 points)
from sklearn.neural_network import MLPClassifier

# a. train a neural network, choosing a network topology of your choice
nn1 = MLPClassifier(hidden_layer_sizes=(32, 16), random_state=1234)
nn1.fit(X_train, y_train)

# b. test and evaluate
y_pred_nn1 = nn1.predict(X_test)

# c. train a second network with a different topology and different settings
nn2 = MLPClassifier(hidden_layer_sizes=(64, 32, 16), activation='tanh', random_state=123
nn2.fit(X_train, y_train)

# d. test and evaluate
y_pred_nn2 = nn2.predict(X_test)

# e. compare the two models and why you think the performance was same/different
print("Neural Network 1 classification report:")
print(classification_report(y_test, y_pred_nn1, zero_division=0))
print("Neural Network 2 classification report:")
print(classification_report(y_test, y_pred_nn2, zero_division=0))

# 11. Analysis (15 points)
# a. which algorithm performed better?
# b. compare accuracy, recall and precision metrics by class
# c. give your analysis of why the better-performing algorithm might have outperformed t
# d. write a couple of sentences comparing your experiences using R versus sklearn. Feel

#a. The Decision Tree algorithm performed better among the models tested. The Decision T
#   the Logistic Regression has an accuracy of 0.86. Both Neural Networks performed poor


#b. Comparing accuracy, recall, and precision metrics by class:

#Logistic Regression:
#Class 0 (low mpg): precision 0.98, recall 0.80
#Class 1 (high mpg): precision 0.73, recall 0.96
#Decision Tree:
#Class 0 (low mpg): precision 0.92, recall 0.90
#Class 1 (high mpg): precision 0.83, recall 0.86

#c. The Decision Tree might have outperformed the Logistic Regression because it can cap
#interactions between features. Decision Trees can also handle non-linear decision bound
#for this particular dataset. On the other hand, Logistic Regression is a linear model,
#model more complex patterns.

#d. Comparing experiences using R versus sklearn (Python):

#R has a more compact and expressive syntax for working with data, which some users migh
#ecosystem is also tailored specifically for statistical modeling and data analysis.
#Sklearn (Python) offers a more general-purpose programming language, which can be a str
#on the user's background and preferences. Python's extensive library ecosystem and skle
```

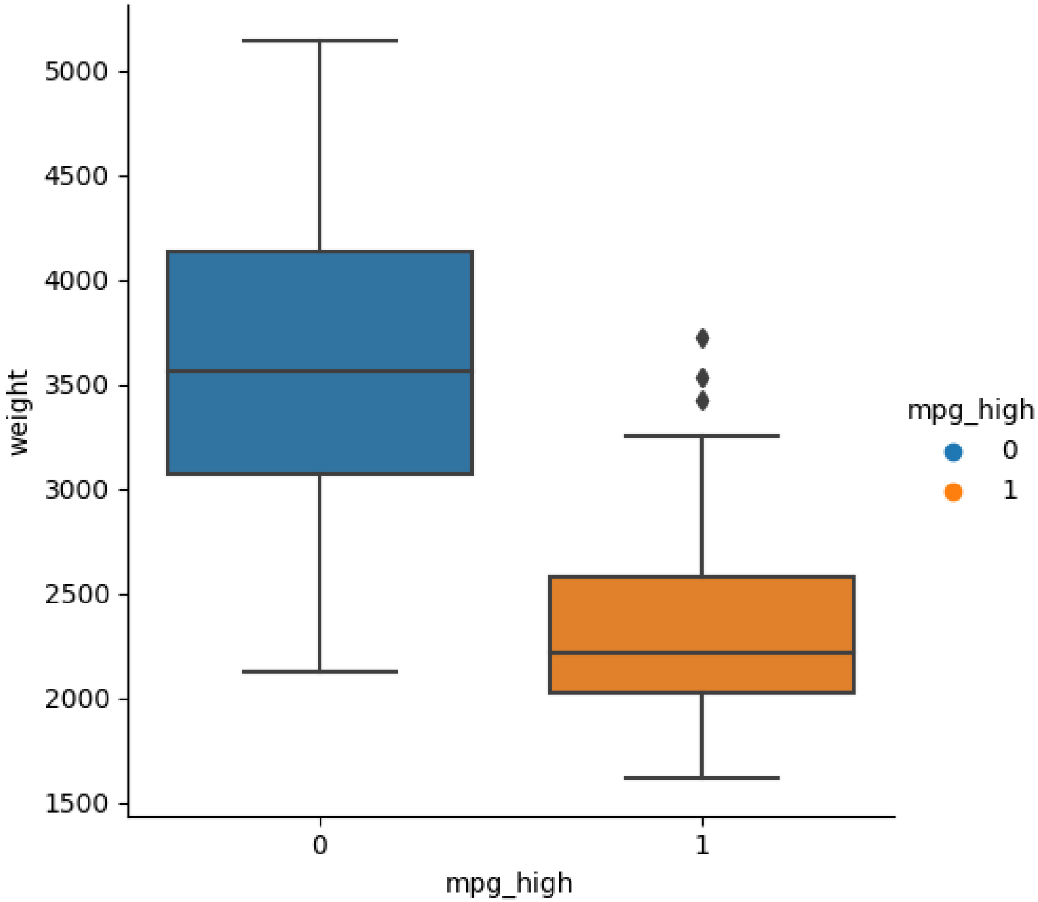
```
#great choice for machine learning tasks, as it allows for easy integration with other l
#visualization, and deployment.
#Both R and sklearn have their strengths and weaknesses, and the choice of which to use
#background, and the specific task at hand. Some users might prefer R for its statistica
#while others might find Python and sklearn more versatile and easier to integrate into
```

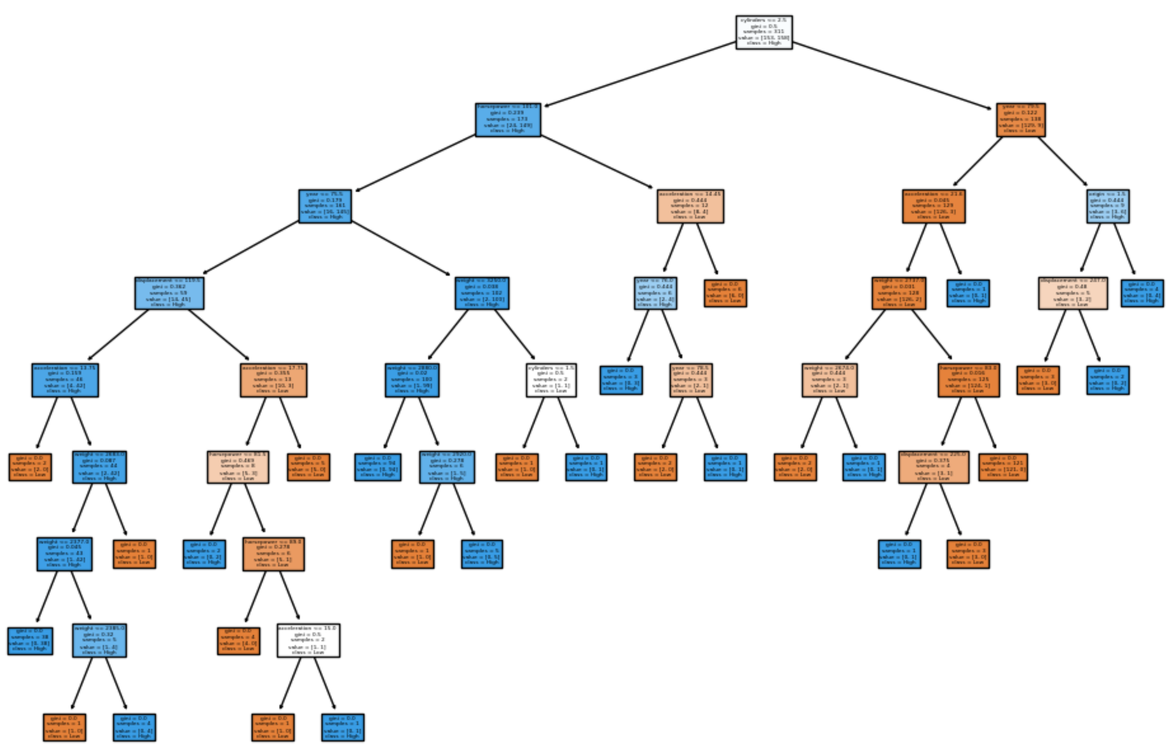
```
    mpg  cylinders  displacement  horsepower  weight  acceleration  year  
0  18.0          8         307.0         130    3504          12.0  70.0  \
1  15.0          8         350.0         165    3693          11.5  70.0   
2  18.0          8         318.0         150    3436          11.0  70.0   
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
(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  
Train dimensions: (311, 7)
Test dimensions: (78, 7)
```

```
Logistic Regression 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

Decision Tree classification report:
              precision    recall  f1-score   support

           0       0.90      0.92      0.91        50
           1       0.85      0.82      0.84        28

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

```
C:\Users\mattm\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0
\LocalCache\local-packages\Python310\site-packages\sklearn\linear_model\_logistic.py:45
8: 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(
```




5000
4500
4000
3500
3000
2500
2000
1500
weight
0
1
mpg_high
mpg_high
0
1



```
Neural Network 1 classification report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        50
           1       0.36      1.00      0.53        28

    accuracy                           0.36        78
   macro avg       0.18      0.50      0.26        78
weighted avg       0.13      0.36      0.19        78

Neural Network 2 classification report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        50
           1       0.36      1.00      0.53        28

    accuracy                           0.36        78
   macro avg       0.18      0.50      0.26        78
weighted avg       0.13      0.36      0.19        78
```

11. Analysis (15 points)

a. which algorithm performed better?

b. compare accuracy, recall and precision metrics by class

c. give your analysis of why the better-performing algorithm might have outperformed the other

d. write a couple of sentences comparing your experiences using R versus sklearn. Feel free to express strong preferences.

a. The Decision Tree algorithm performed better among the models tested. The Decision Tree has an accuracy of 0.88, while

the Logistic Regression has an accuracy of 0.86. Both Neural Networks performed poorly with an accuracy of 0.36.

b. Comparing accuracy, recall, and precision metrics by class:

Logistic Regression:

Class 0 (low mpg): precision 0.98, recall 0.80

Class 1 (high mpg): precision 0.73, recall 0.96

Decision Tree:

Class 0 (low mpg): precision 0.92, recall 0.90

Class 1 (high mpg): precision 0.83, recall 0.86

c. The Decision Tree might have outperformed the Logistic Regression because it can capture more complex relationships and

interactions between features. Decision Trees can also handle non-linear decision boundaries, which might be better suited

for this particular dataset. On the other hand, Logistic Regression is a linear model, which could limit its ability to

model more complex patterns.

d. Comparing experiences using R versus sklearn (Python):

R has a more compact and expressive syntax for working with data, which some users might find more convenient. R's

ecosystem is also tailored specifically for statistical modeling and data analysis.

Sklearn (Python) offers a more general-purpose programming language, which can be a strength or a weakness, depending

on the user's background and preferences. Python's extensive library ecosystem and sklearn's consistent API make it a

great choice for machine learning tasks, as it allows for easy integration with other libraries for data preprocessing,

visualization, and deployment.

Both R and sklearn have their strengths and weaknesses, and the choice of which to use depends on personal preference,

background, and the specific task at hand. Some users might prefer R for its statistical focus and more expressive syntax,

while others might find Python and sklearn more versatile and easier to integrate into a broader software ecosystem.