In [1]:
```python
print("Hello World!")
```

```
Hello World!
```

In [2]:
```python
# 1.a
import pandas as pd
spreadsheet = pd.read_csv('C:/Users/colto/Downloads/auto.csv')
```

In [3]:
```python
# 1.b
spreadsheet.head()
```

Out[3]:

	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	ford torino

In [4]:
```python
# 1.c
print("Rows: ", len(spreadsheet))
print("Columns: ", len(spreadsheet.columns))
```

```
Rows:  392
Columns:  9
```

In [5]:
```python
# 2.a
spreadsheet['mpg'].describe()
spreadsheet['weight'].describe()
spreadsheet['year'].describe()
```

Out[5]:
```
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 [6]:
```python
# 2.b Range
ss1 = spreadsheet.iloc[:, 0:8]
print(ss1.max() - ss1.min())
```

```
mpg               37.6
cylinders          5.0
displacement     387.0
horsepower       184.0
weight          3527.0
acceleration      16.8
year              12.0
origin             2.0
dtype: float64
```

In [7]:
```python
# 2.b Range
print(ss1.max() - ss1.min())
```

```
mpg               37.6
cylinders          5.0
displacement     387.0
horsepower       184.0
weight          3527.0
acceleration      16.8
year              12.0
origin             2.0
dtype: float64
```

In [8]:
```python
# 3.a
spreadsheet.dtypes
```

Out[8]:
```
mpg             float64
cylinders         int64
displacement    float64
horsepower        int64
weight            int64
acceleration    float64
year            float64
origin            int64
name             object
dtype: object
```

In [9]:
```python
# 3.b
spreadsheet['cylinders'] = spreadsheet['cylinders'].astype('category').cat.codes
```

In [10]:
```python
# 3.c
spreadsheet['origin'] = spreadsheet['origin'].astype('category')
```

In [11]:
```python
# 3.d
spreadsheet.dtypes[['cylinders','origin']]
```

Out[11]:
```
cylinders        int8
origin       category
dtype: object
```

In [12]:
```python
# 4.a
spreadsheet = spreadsheet.dropna()
```

In [13]:
```python
# 4.b
print("Rows: ", len(spreadsheet)) # Originally 392
print("Columns: ", len(spreadsheet.columns)) # Originally 9
```

```
Rows:  389
Columns:  9
```

In [14]:
```python
# 5.a
avg_mpg = spreadsheet['mpg'].mean()
spreadsheet['mpg_high'] = 0
spreadsheet.loc[spreadsheet['mpg'] > avg_mpg, 'mpg_high'] = 1
```

In [15]:
```python
# 5.b
spreadsheet = spreadsheet.drop(columns=['mpg','name'])
```

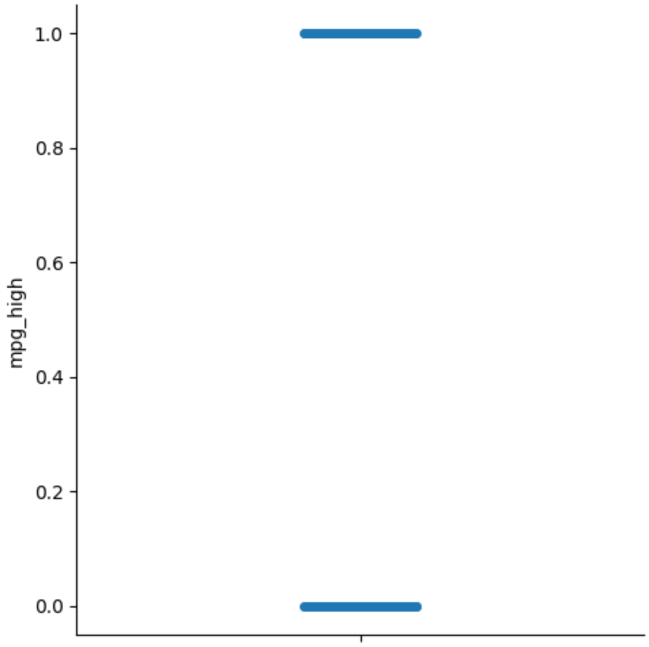
In [16]:
```python
# 5.c
spreadsheet.head()
```

Out[16]:

	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

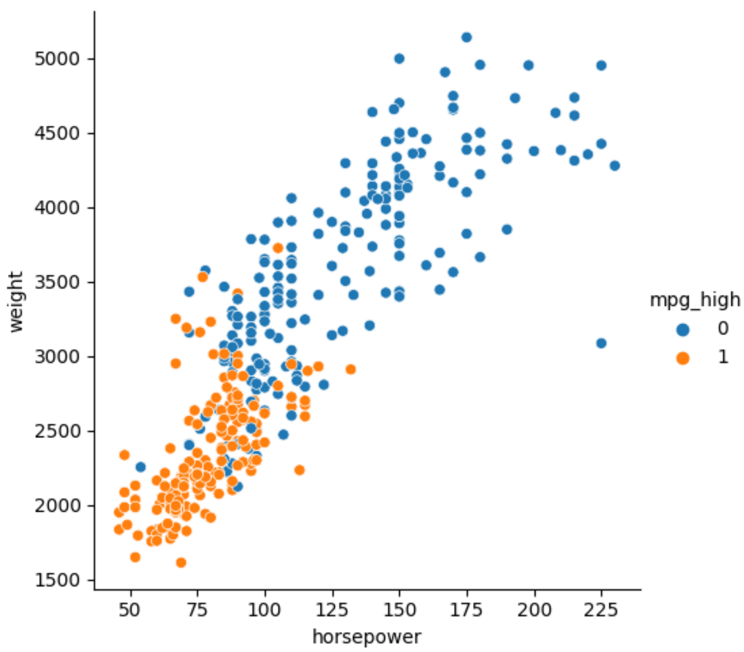
In [17]:
```python
# 6.a
import seaborn as sns
sns.catplot(spreadsheet['mpg_high'])
```

Out[17]: <seaborn.axisgrid.FacetGrid at 0x278173bee50>



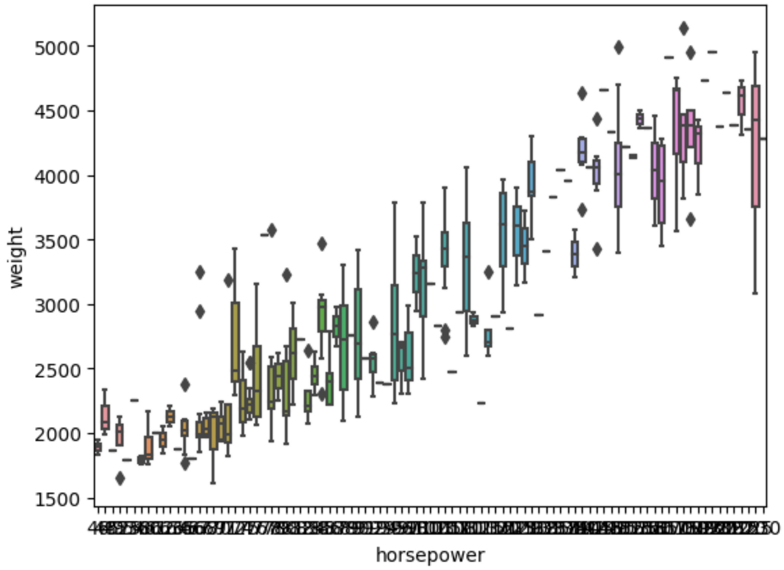
In [18]:
```python
# 6.b
sns.relplot(x='horsepower', y='weight', hue='mpg_high', data=spreadsheet)
```

Out[18]: <seaborn.axisgrid.FacetGrid at 0x278174fd3d0>



In [19]:
```python
# 6.c
sns.boxplot(x='horsepower', y='weight', data=spreadsheet)
```

Out[19]: <Axes: xlabel='horsepower', ylabel='weight'>



In [20]:
```python
# 6.d
# The graph from 6.a shows that only the values 0 and 1 exist inside the mpg_high category.
# The graph from 6.b shows that observations with a mpg_high value of 1 tend to have low horsepower and weight compared to obse
# The graph from 6.c shows that outliers tend to have high weight and low horsepower rather than low weight and high horsepower
```

In [21]:
```python
# 7.a
split = 0.2
```

In [22]:
```python
# 7.b
seed = 1234
```

In [23]:
```python
# 7.c
from sklearn.model_selection import train_test_split

data = spreadsheet
x = data.iloc[:, 0:7] # x = feature matrix
y = data['mpg_high'] # y = target variable
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=split, random_state=seed)
```

In [24]:
```python
# 7.d
print("x_train shape:", x_train.shape)
print("x_test shape:", x_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
x_train shape: (311, 7)
x_test shape: (78, 7)
y_train shape: (311,)
y_test shape: (78,)
```

In [25]:
```python
# 8.a
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(solver='lbfgs', max_iter=1000)
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
```

In [26]:
```
# 8.b
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1_score = f1_score(y_test, y_pred)

print("Accuracy: {:.2f}".format(accuracy))
print("Precision: {:.2f}".format(precision))
print("Recall: {:.2f}".format(recall))
print("F1 Score: {:.2f}".format(f1_score))
```

```
Accuracy: 0.90
Precision: 0.78
Recall: 1.00
F1 Score: 0.88
```

In [27]:
```
# 8.c
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.84      0.91        50
           1       0.78      1.00      0.88        28

    accuracy                           0.90        78
   macro avg       0.89      0.92      0.89        78
weighted avg       0.92      0.90      0.90        78
```

In [28]:
```
# 9.a
from sklearn.tree import DecisionTreeClassifier

dt_model = DecisionTreeClassifier()
dt_model.fit(x_train, y_train)
```

Out[28]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [29]:
```
# 9.b
y_pred = dt_model.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: {:.2f}".format(accuracy))
```

```
Accuracy: 0.90
```

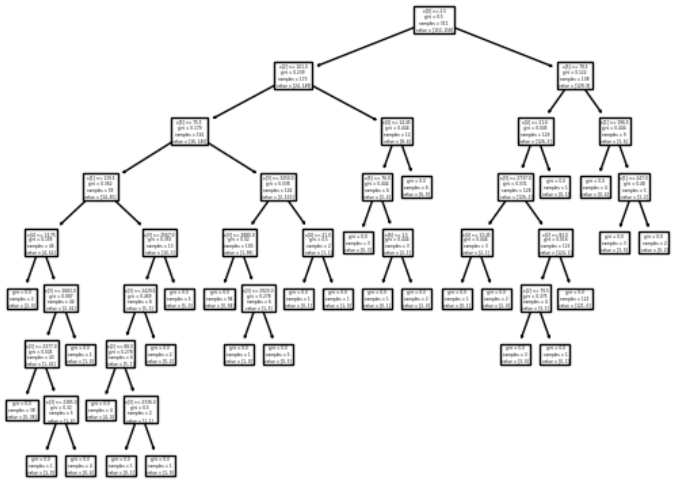
In [30]:
```
# 9.c
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.86      0.91        50
           1       0.79      0.96      0.87        28

    accuracy                           0.90        78
   macro avg       0.89      0.91      0.89        78
weighted avg       0.91      0.90      0.90        78
```

In [31]:
```
# 9.d
from sklearn import tree
import matplotlib.pyplot as plt
tree.plot_tree(dt_model)
plt.show()
```



In [32]:
```
# 10.a
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler1 = StandardScaler()
scaler2 = StandardScaler()
scaler.fit(x_train)
scaler1.fit(x_test)

x_train_scaled = scaler.transform(x_train)
x_test_scaled = scaler1.transform(x_test)

regr = MLPRegressor(hidden_layer_sizes=(6,3), max_iter=1000, random_state=seed)
regr.fit(x_train_scaled, y_train)
regr.fit(x_test_scaled, y_test)
```

Out[32]:
```
MLPRegressor
MLPRegressor(hidden_layer_sizes=(6, 3), max_iter=1000, random_state=1234)
```

In [33]:
```
# 10.b
from sklearn.metrics import mean_squared_error, r2_score

y_pred_regr = regr.predict(x_test_scaled) # make predictions

print('mse =', mean_squared_error(y_test, y_pred_regr))
print('correlation =', r2_score(y_test, y_pred_regr))
```

```
mse = 0.05527247016944208
correlation = 0.7598016367779388
```

In [34]:
```
# 10.c

scaler = StandardScaler()
scaler1 = StandardScaler()
scaler.fit(x_train)
scaler1.fit(x_test)

x_train_scaled = scaler.transform(x_train)
x_test_scaled = scaler1.transform(x_test)

regr = MLPRegressor(hidden_layer_sizes=(12,6), max_iter=1000, random_state=seed, solver='adam') # different topology, differen
regr.fit(x_train_scaled, y_train)
regr.fit(x_test_scaled, y_test)
```

Out[34]:
```
MLPRegressor
MLPRegressor(hidden_layer_sizes=(12, 6), max_iter=1000, random_state=1234)
```

In [35]:
```
# 10.d
y_pred_regr = regr.predict(x_test_scaled) # make predictions

print('mse =', mean_squared_error(y_test, y_pred_regr))
print('correlation =', r2_score(y_test, y_pred_regr))
```

```
mse = 0.04772849165185324
correlation = 0.7925856119929463
```

In [36]:
```
# 10.e
# After playing around with different settings, it was difficult to get a healthier mse than the first model.
# But if I increase the size of the topology while retaining the same 2:1 node ratio, I can get an MSE lower than model 1.
# It makes sense that both models perform relatively the same because their toplogy & settings are similar.
# However, as I increase the number of nodes, I risk overfitting the data.
```

In [37]:
```
# 11.a
# Neural networks took the longest to execute. Logistic regression executed just about as fast as the decision tree.
# However, decision trees looks like it reports better metrics than the decision tree. (Investigated in 11.b)
```

In [38]:
```
# 11.b
# Logistic Regression
#               precision    recall  f1-score   support
#
#            0       1.00      0.84      0.91        50
#            1       0.78      1.00      0.88        28
#
#     accuracy                           0.90        78
#    macro avg       0.89      0.92      0.89        78
#weighted avg       0.92      0.90      0.90        78
# Decision Tree
#               precision    recall  f1-score   support
#
#            0       0.94      0.92      0.93        50
#            1       0.86      0.89      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
#
# It's difficult to say which performed better because their numbers are similar.
# But it looks like the decision tree reports better metrics for f1-score while precision and recall are relatively the same.
```

In [39]:
```
# 11.c
# In the areas where decision trees might have outperformed logistic regression, it can be partially credited to it's robustnes
# Alternatively, there may be some interaction between features which decision trees are great at capturing.
# On the other hand, there might not have been much interaction so logistic regression was able to keep up with decision trees.
```

In [40]:
```
# 11.d
# sklearn was nice to pick up since I'm familiar with python, but my kernel had crashed a couple times so far.
# It's a little troublesome since I wouldn't immediately realize that it's stuck or crashed.
# I still prefer sklearn, I especially like having my development environment in my browser so it's easier to look up informati
```