```
import pandas as pd
df = pd.read_csv("Auto.csv")
print(df.head())
print("\nDimensions of data frame:", 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

Dimensions of data frame: (392, 9)
```

```
# range 9 -> 46.6, average -> 23.5
print(df["mpg"].describe(), "\n")
# range 1613 -> 5140, average -> 2970.30
print(df["weight"].describe(), "\n")
# range 70 -> 82, average -> 76.0
print(df["year"].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

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

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
```

```
print(df.dtypes, "\n")
df.cylinders = df.cylinders.astype("category").cat.codes
df.origin = df.origin.astype("category")
print(df.dtypes)
```

```
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
```

```
df = df.dropna()
print("New dimensions output:", df.shape)
```

```
New dimensions output: (389, 9)
```

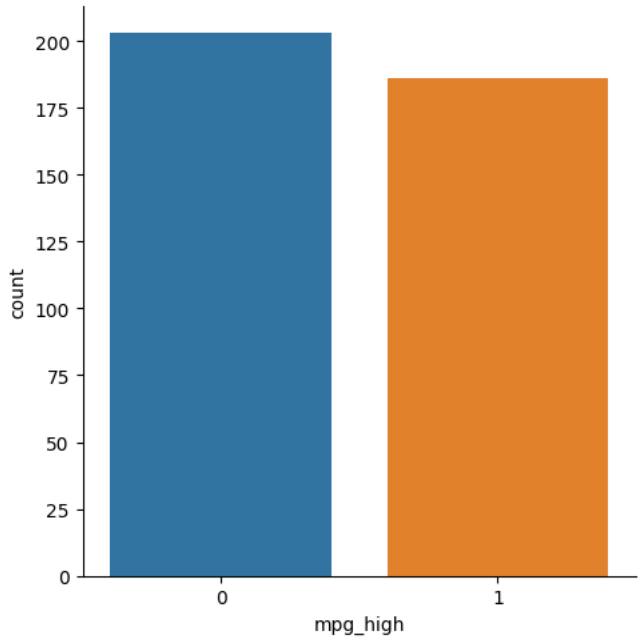
```
df["mpg_high"] = (df.mpg > df.mpg.mean()).astype(int)
df.mpg_high = df.mpg_high.astype("category")
df = df.drop(columns=["mpg", "name"])
print(df.head())
```

```
   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
```

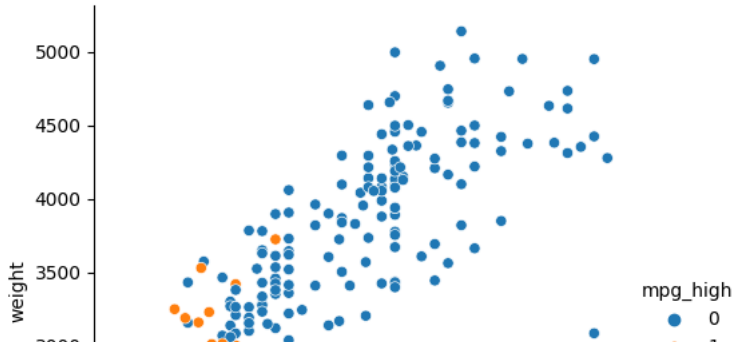
```
import seaborn as sns
# most cars mpg is less than the average of all cars in the dataframe
sns.catplot(x="mpg_high", kind="count", data=df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7ff9d1462cd0>
```



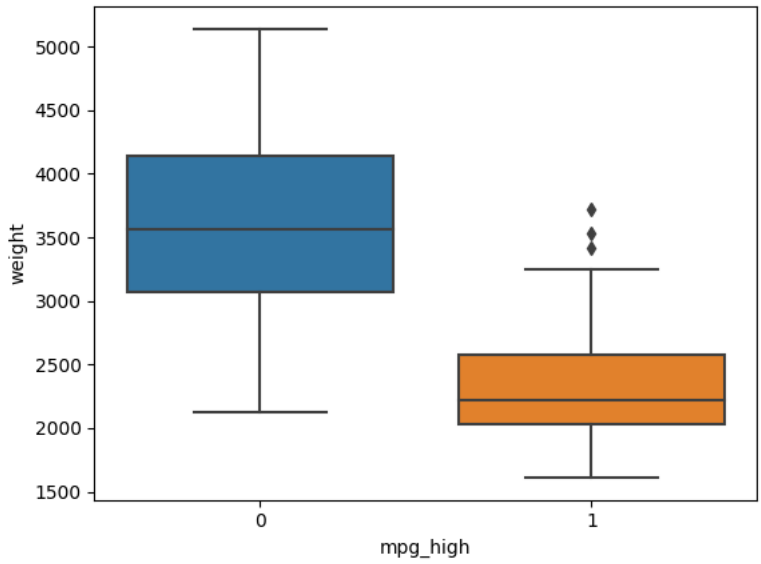

```
# high mpg cars -> lower horsepower and lower weight
sns.relplot(x="horsepower", y="weight", hue="mpg_high", data=df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7ff9744df310>
```




```
# high mpg cars -> lower weight
sns.boxplot(x="mpg_high", y="weight", data=df)
```

```
<Axes: xlabel='mpg_high', ylabel='weight'>
```




```
from sklearn.model_selection import train_test_split
X = df.loc[:, ["cylinders", "displacement", "horsepower", "weight", "acceleration", "year", "origin"]]
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 Set Dimensions:", X_train.shape)
print("Test Set Dimensions:", X_test.shape)
```

```
Train Set Dimensions: (311, 7)
Test Set Dimensions: (78, 7)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
model = LogisticRegression(solver='lbfgs', max_iter=500000)
model.fit(X_train, y_train)
pred = model.predict(X_test)
print('accuracy score: ', accuracy_score(y_test, pred))
print(classification_report(y_test, pred))
```

```
accuracy score:  0.8589743589743589
              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
```

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(random_state=1234)
model.fit(X_train, y_train)
pred = model.predict(X_test)
```

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

```
accuracy score:  0.9230769230769231
              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
```

```
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
from sklearn.metrics import classification_report
clf = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5, 2), max_iter=1500, random_state=1234)
clf.fit(X_train_scaled, y_train)
pred = clf.predict(X_test_scaled)
print('accuracy = ', accuracy_score(y_test, pred))
print(classification_report(y_test, pred))
```

```
accuracy =  0.8589743589743589
              precision    recall  f1-score   support

           0       0.93      0.84      0.88        50
           1       0.76      0.89      0.82        28

    accuracy                           0.86        78
   macro avg       0.85      0.87      0.85        78
weighted avg       0.87      0.86      0.86        78
```

```
clf = MLPClassifier(solver='sgd', hidden_layer_sizes=(3,), max_iter=3000, random_state=1234)
clf.fit(X_train_scaled, y_train)
pred = clf.predict(X_test_scaled)
print('accuracy = ', accuracy_score(y_test, pred))
print(classification_report(y_test, pred))
```

```
accuracy =  0.8333333333333334
              precision    recall  f1-score   support

           0       0.93      0.80      0.86        50
           1       0.71      0.89      0.79        28

    accuracy                           0.83        78
   macro avg       0.82      0.85      0.83        78
weighted avg       0.85      0.83      0.84        78
```

I trained the neural network with two different network topologies. One was using a single hidden layer and the other was using multiple hidden layers. I increased the max iterations for the single hidden layer to maybe account for the fact we had only one hidden layer. According to the classification reports, the multiple hidden layer model performed better than the single layer MLP classifier. The performance might have been different because since one of the network toplogies is using more layers, the multiple hidden layer toplogy might find more complex relationships within the features and target variable than the single layer toplogy.

The decision tree classifier performed better than all the other algorithms in regards to accuracy. For Class 0, the logistic regression model had the highest precision and the decision tree classifier had the highest recall. For Class 1, the decision tree classifier had the highest precision while the logistic regression model had the highest recall. For the neural network models, the multiple hidden layer NLP classifier had a higher recall but a lower precision than the single layer. For Class 1, the multiple hidden layer topology had a higher precision and had the same recall score. Overall, the decision tree classifier might have outperformed other models due its ability to find complex relationships within the features and target variable versus the logistic regression model. Also, we normalized our data before using the neural network models so this might be why the decision tree classifier outperformed those models as well. I prefer using sklearn over R. With Python, it only requires a few lines of code to clean, normalize, and scale data compared to doing this with R. When it comes to more advanced algorithms like training neural networks or decision tree classifiers, we can just instantiate one instance from the sklearn module and fill in the parameters easier than R.

✓ 0s completed at 6:04 PM