Zachary Tarell - zjt170000

Homework 7: Machine Learning with Python

1. Read the Auto data

In [77]:
```
# a) use pandas to read the data
import numpy as np
import pandas as pd
df = pd.read_csv('Auto.csv')
# b) print the first few rows
print(df.head())
# c) print the dimensions of the data
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)
```

2. Data Exploration

In [78]:
```
# a) use describe() on the mpg, weight, and year columns
print('\nDescribe mpg, weight, and year:\n', df.loc[:, ['mpg', 'weight', 'year']].d
escribe())
# b) write comments indicating the range and average of each column
print('\nThe range of the cars YEAR in the data is from 1970-1982 ({}) years.'.form
at(13))
print('The range of the cars WEIGHT in the data is 1,613-5,140 lbs.')
print('The range of the cars MPG in the data is 9-46 miles per gallon.')
print('So, average MPG = 23.45, WEIGHT = 2,977.58 lbs, and YEAR = 1976\n')
```

```
Describe mpg, weight, and year:
               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

The range of the cars YEAR in the data is from 1970-1982 (13) years.
The range of the cars WEIGHT in the data is 1,613-5,140 lbs.
The range of the cars MPG in the data is 9-46 miles per gallon.
So, average MPG = 23.45, WEIGHT = 2,977.58 lbs, and YEAR = 1976
```

3. Explore Data Types

In [79]:
```
# a) check the data types of all columns
df.dtypes
print(df.dtypes, "\n")
# b) change the cylinders column to categorical (use cat.codes)
df.cylinders = df.cylinders.astype('category').cat.codes
# c) change the origin column to categorical (don't use cat.codes)
df.origin = df.origin.astype('category')
# d) verify the changes with the dtypes attribute
df.dtypes
```

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

Out[79]:
```
mpg              float64
cylinders           int8
displacement     float64
horsepower         int64
weight             int64
acceleration     float64
year             float64
origin          category
name              object
dtype: object
```

4. Deal with NAs

In [80]:
```
# a) delete rows with NAs
df = df.dropna()
# b) print the new dimensions
print('\nDimensions of data frame:', df.shape)
```

```
Dimensions of data frame: (389, 9)
```

5. Modify Columns

In [81]:
```
# a) make a new column, mpg_high, which is categorical: the column == 1 if mpg > av
erage mpg, else == 0
df['mpg_high'] = np.where(df['mpg'] > df['mpg'].mean(), '1', '0')
df['mpg_high'] = df['mpg_high'].astype('category').cat.codes
# b) delete the mpg and name columns
df = df.drop(columns=['mpg', 'name'])
# c) print the first few rows of the modified data frame
print(df.head())
```

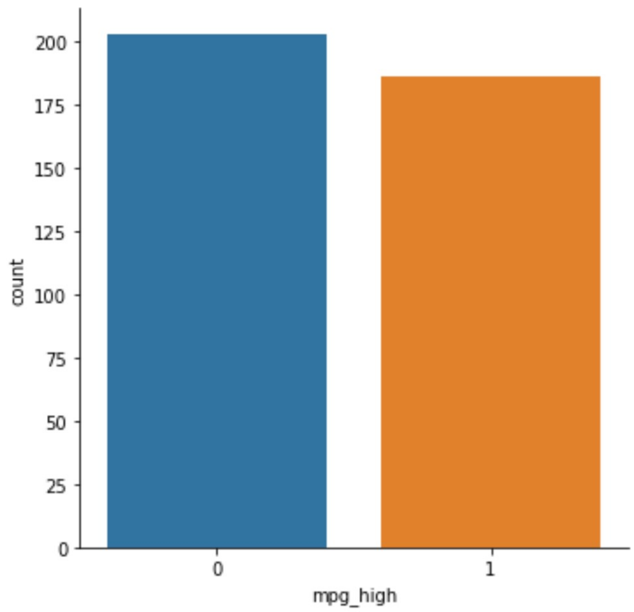
```
   cylinders  displacement  horsepower  weight  acceleration  year  origin  \
0          4         307.0         130    3504          12.0  70.0       1
1          4         350.0         165    3693          11.5  70.0       1
2          4         318.0         150    3436          11.0  70.0       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
```

6. Data Exploration with Graphs

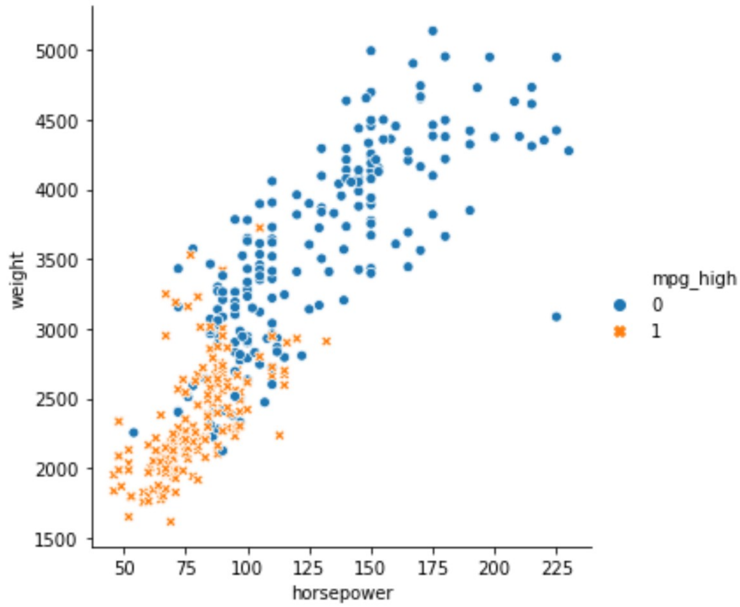
In [82]:
```
# a) seaborn catplot on the mpg_high column
# for each graph, write a comment indicating one thing you learned about the data f
rom the graph
import seaborn as sb
from sklearn import datasets
sb.catplot(x='mpg_high', kind = 'count', data=df)
print('\nThis catplot on the Auto data shows that the mpg_high column is split pret
ty equally.')
print('There is a little more cars that get over the average rate of mpg then under
it.')
```

```
This catplot on the Auto data shows that the mpg_high column is split pretty equ
ally.
There is a little more cars that get over the average rate of mpg then under it.
```



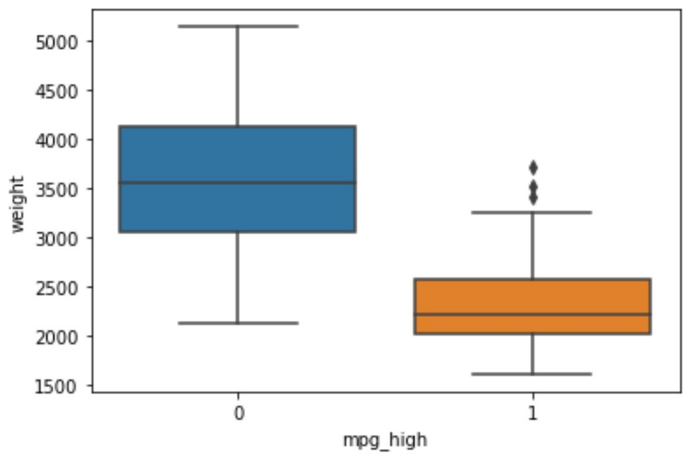
In [83]:
```
# b) seaborn relplot with horsepower on the x axis, weight on the y axis, setting h
ue or style to mpg_high
sb.relplot(x='horsepower', y='weight', data=df, hue=df.mpg_high, style=df.mpg_high)
print('\nThe relplot shows the plotting of 2 quantitative arrays - horsepower and w
eight - according to the mpg_high class.')
print('It shows those cars with higher weight and horsepower have worse rates of mp
g, and are related linearly.')
```

```
The relplot shows the plotting of 2 quantitative arrays - horsepower and weight
- according to the mpg_high class.
It shows those cars with higher weight and horsepower have worse rates of mpg, a
nd are related linearly.
```



In [84]:
```
# c) seaborn boxplot with mpg_high on the x axis and weight on the y axis
sb.boxplot('mpg_high', y='weight', data=df)
print('\nThe boxplot is an easy way to tell where the median and most of the data l
ay - especially the outliers.')
print('This shows that the median weight for cars with better gas mileage is about
3500 lbs with a high range.')
print('And, the median for worse gas mileage is around 2300 lbs with a few outliers
and a much smaller range.')
```

```
The boxplot is an easy way to tell where the median and most of the data lay - e
specially the outliers.
This shows that the median weight for cars with better gas mileage is about 3500
lbs with a high range.
And, the median for worse gas mileage is around 2300 lbs with a few outliers and
a much smaller range.
```



7. Train/Test Split

In [85]:
```
from sklearn.model_selection import train_test_split
# a) 80/20
# b) use seed 1234 so we all get the same results
# c) train /test X data frames consists of all remaining columns except mpg_high
X = df.iloc[:, 0:6]
y = df.iloc[:, 7]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
te=1234)
# d) print the dimensions of train and test
print('train size:', X_train.shape)
print('test size:', X_test.shape)
```

```
train size: (311, 6)
test size: (78, 6)
```

8. Logistic Regression

In [86]:
```
from sklearn.linear_model import LogisticRegression
# a) train a logistic regression model using solver lbfgs
clf = LogisticRegression(solver='lbfgs', max_iter=500)
clf.fit(X_train, y_train)
clf.score(X_train, y_train)
```

Out[86]: 0.8938906752411575

In [87]:
```
# b) test and evaluate
# make predictions
pred = clf.predict(X_test)
# Evaluate
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred))
print('recall score: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))
```

```
accuracy score:  0.8974358974358975
precision score:  0.7941176470588235
recall score:  0.9642857142857143
f1 score:  0.8709677419354839
```

In [88]:
```
# c) print metrics using the classification report
# Confusion Matrix
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, pred)
```

Out[88]:
```
array([[43,  7],
       [ 1, 27]], dtype=int64)
```

In [89]:
```
# Classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, 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
```

9. Desicion Tree

In [90]:
```
# a) train a decision tree
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
clf.score(X_train, y_train)
```

Out[90]:
```
1.0
```

In [91]:
```
# b) test and evaluate
# make predictions
pred = clf.predict(X_test)
# Evaluate
print('accuracy score: ', accuracy_score(y_test, pred))
print('precision score: ', precision_score(y_test, pred))
print('recall score: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))
```

```
accuracy score:  0.9230769230769231
precision score:  0.84375
recall score:  0.9642857142857143
f1 score:  0.8999999999999999
```

In [92]:
```
# c) print the classification report metrics
# confusion matrix
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, pred)
```

Out[92]:
```
array([[45,  5],
       [ 1, 27]], dtype=int64)
```

In [93]:
```
# Classification report
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.90      0.94        50
           1       0.84      0.96      0.90        28

    accuracy                           0.92        78
   macro avg       0.91      0.93      0.92        78
weighted avg       0.93      0.92      0.92        78
```

In [94]:
```
# d) plot the tree (optional)
#from sklearn import tree
#tree.plot_tree(clf.fit(X, y))
```

10. Analysis

a)

```
The algorithm that performed better was the Desicion Tree with an accuracy score of 9
2%.
```

b)

```
For accuracy, recall, and precision: LogReg had .90, .90, .91 as weighted averages
For accuracy, recall, and precision: D-Tree had .92, .92, .93 as weighted averages

The main difference was from the test and pred classsifications. LogReg performed bad
in these categories on the recall test (.86) and precision predictor (.79). But the De
cision Tree had .92 and .87 for those same categories. This is what ultimately lowered
the overall average of LogReg comparatively to the Decision Tree.
```

c)

```
I think that the reason the Desicion Tree outperformed the Logistic Regression algorit
hm was because this was a small data set. Becasue of that, the Decision Tree (which is
prone to overfit) didn't really have to worry to much about that. Also, Logistic Regre
ssion has a linear and single decision boundary while Decision Tree bisects the space
into smaller spaces which help it to outperform on smaller data sets.

A few side notes:
    The DT plot prints out the entire array and then a small picture of a tree afterwa
rd - I think it's the JupyterLab IDE but not sure.
    And for some reason (I couldn't figure out) my confusion matrices print out "dtype
=int64" outside the array - I don't know why.
```