ML with Sklearn
Nikilas John
Professor Mazidi
November 6th, 2022

In [150]:
```
from google.colab import files
uploaded = files.upload()
```

Browse... No files selected.

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Auto.csv to Auto (4).csv
```

Reading in the data

This section will read in the data and then output the first few entries

In [151]:
```
import pandas as pd
df = pd.read_csv('Auto.csv')
df.head()
```

Out[151]:

	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

Displaying the dimensions of the data

This section will display the dimensions of the data which is formatted (rows, columns)

In [152]:
```
df.shape
```

Out[152]:
```
(392, 9)
```

Data exploration section

This section will explore the data

Displaying the statistics on mpg column

Range: 37
Average: 23.445918

In [153]:
```
df.mpg.describe()
```

Out[153]:
```
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
```

Displaying the statistics on weight column

Range: 3527
Average: 2977.584184

In [154]:
```
df.weight.describe()
```

Out[154]:
```
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
```

Displaying the statistics on year column

Range: 12
Average: 76.010256

In [155]:
```
df.year.describe()
```

Out[155]:
```
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
```

Checking the data types of the columns

In [156]:
```
df.dtypes
```

Out[156]:
```
mpg             float64
cylinders         int64
displacement    float64
horsepower        int64
weight            int64
acceleration    float64
year            float64
origin            int64
name             object
dtype: object
```

Converting the cylinders column to categorical data

In [157]:
```
df.cylinders = df.cylinders.astype('category').cat.codes
```

Converting origin column to categorical data (not using cat.codes)

In [158]:
```
df.origin = df.origin.astype('category')
```

Verify Results

In [159]:
```
df.dtypes
```

Out[159]:
```
mpg              float64
cylinders           int8
displacement     float64
horsepower         int64
weight             int64
acceleration     float64
year             float64
origin          category
name              object
dtype: object
```

Deleting the rows with NAs

In [160]:
```
df = df.dropna()
```

Verify Results

Let's see if deleting the NAs worked

In [161]:
```
df.shape
```

Out[161]:
```
(389, 9)
```

Modifying the columns

Time to modify the columns, adding an mpg_high column to replace the mpg column

Creating the mpg_high column

This section will create the mpg_high column and make it categorical

In [162]:
```
def func(x):
    if x > df.mpg.mean():
        return 1
    return 0

df = df.assign(mpg_high=df.mpg.apply(func))
```

Convert the mpg_high column to categorical data

In [163]:
```
df.mpg_high = df.mpg_high.astype('category').cat.codes
```

Delete the mpg and name columns

In [164]:
```
del df['mpg']

del df['name']
```

Verify results

We can see that the mpg and name columns have been deleted, so we are good to go.

In [165]:
```
df.head()
```

Out[165]:

	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

Data exploration with graphs

In this section, we will start exploring the data using the seaborn library

In [166]:
```
import seaborn as sb
```

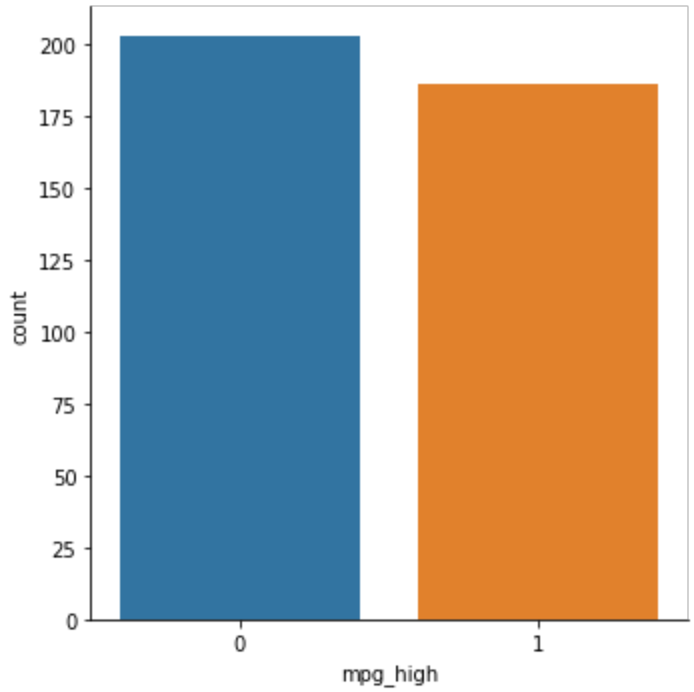
Seaborn Catplot

Here we will use category plot to plot the mpg_high column of df

One thing I learned from this graph is that there are slightly more cars that are not "fuel efficient" by the standard we created when we made the mpg_high column

In [167]:
```
sb.catplot(x="mpg_high", kind="count", data=df)
```

Out[167]: <seaborn.axisgrid.FacetGrid at 0x7f7ab70eb090>



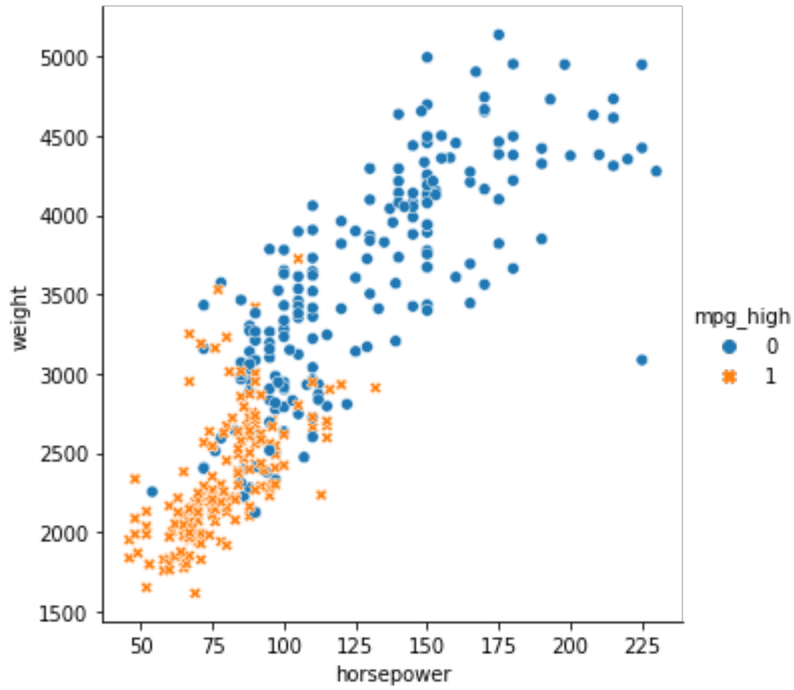
Seaborn Relplot

Here we will use relation plot to plot the relationship between horsepower and weight, using mpg_high as the style and hue

One observation I can make is that the cars with the lowest horsepowers and weights are clustered closer than the one with higher horsepower and weight

In [168]:
```
sb.relplot(x="horsepower", y="weight", data=df, hue=df.mpg_high, style=df.m
pg_high)
```

Out[168]:
```
<seaborn.axisgrid.FacetGrid at 0x7f7ab6eb6a90>
```



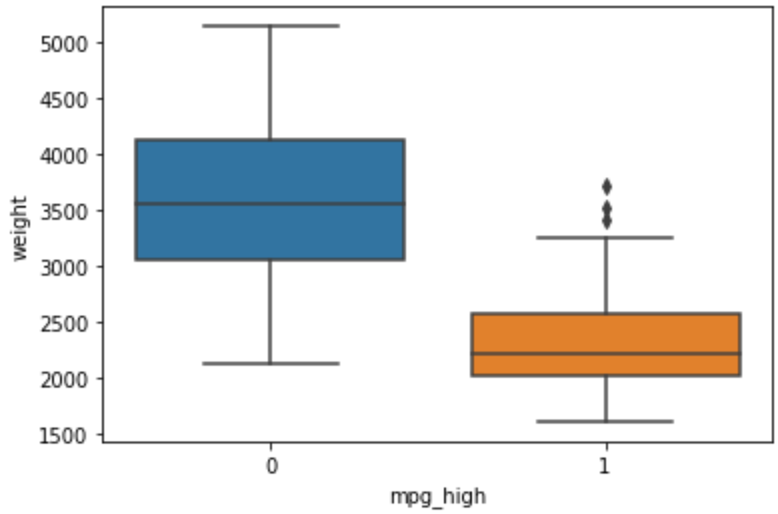
Seaborn Boxplot

Here we will use the box plot to plot the mpg_high and weight on the x and y axis respectively

One thing I learned about this graph is that higher weight means that the car is way more liekly to have lower than average mpg, while if your car has lower weight, you are significantly more likely to have a car with better than average mpg

In [169]:
```
sb.boxplot(x="mpg_high", y="weight", data=df)
```

Out[169]: <matplotlib.axes._subplots.AxesSubplot at 0x7f7ab6e264d0>



Train/Test Split

Time to split the data into training and testing data using the train_test_split library

In [170]:
```
from sklearn.model_selection import train_test_split

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, ra
ndom_state=1234)

print("train size:", X_train.shape)
print("test size:", X_test.shape)
```

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

Logistic Regression

In this section we will do logistic regression in sklearn

In [171]:
```
from sklearn.linear_model import LogisticRegression
```

Training

In [172]:
```
clf = LogisticRegression()
clf.fit(X_train, y_train)
clf.score(X_train, y_train)
```

Out[172]: 0.9035369774919614

Testing

In [173]:
```
pred = clf.predict(X_test)
```

Evaluation

Here we will evaluate the predictions using the classification report

In [174]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, pred))
```

```
              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

Here we will create a decision tree for classification

In [175]:
```
from sklearn.tree import DecisionTreeClassifier
```

Training

In [187]:
```
clf = DecisionTreeClassifier(random_state=1234)
clf.fit(X_train, y_train)
```

Out[187]: DecisionTreeClassifier(random_state=1234)

Testing

In [188]:
```
pred = clf.predict(X_test)
```

Evaluation

In [189]:
```
print(classification_report(y_test, pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.90      0.93        50
           1       0.84      0.93      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
```

Neural Network

In this section, we will create a neural network with the data

In [179]:
```
from sklearn import preprocessing
```

Preprocess the data

In [180]:
```
scaler = preprocessing.StandardScaler().fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

Training

In [181]:
```
from sklearn.neural_network import MLPRegressor
regr = MLPRegressor(solver='lbfgs', hidden_layer_sizes=(6, 3), max_iter=80
0, random_state=1234)
regr.fit(X_train, y_train)
```

Out[181]:
```
MLPRegressor(hidden_layer_sizes=(6, 3), max_iter=800, random_state=1234,
             solver='lbfgs')
```

Testing

In [182]:
```
y_pred = regr.predict(X_test)
```

Evaluation

In [183]:
```
from sklearn.metrics import mean_squared_error, r2_score
print('mse=', mean_squared_error(y_test, y_pred))
print('correlation=', r2_score(y_test, y_pred))
```

```
mse= 0.2524058375197891
correlation= -0.0968836539074267
```

Training using different topology and settings

In [184]:
```
regr = MLPRegressor(hidden_layer_sizes=(7, 4), max_iter=800, random_state=1
234)
regr.fit(X_train, y_train)
```

Out[184]:
```
MLPRegressor(hidden_layer_sizes=(7, 4), max_iter=800, random_state=1234)
```

Testing new network

In [185]:
```
y_pred = regr.predict(X_test)
```

Evaluating neural network

In [186]:
```
print('mse=', mean_squared_error(y_test, y_pred))
print('correlation=', r2_score(y_test, y_pred))
```

```
mse= 0.10592523380437101
correlation= 0.539679198238719
```

Comparison of the networks

The network with the hidden layer sizes set to (7, 4) seems to have performed the best. I tested a lot of other networks but none of them had a MSE comparable to the second one. I suspect the reason why this happened was because of the number of nodes on the hidden layers. When I changed the number of iterations, the MSE and correlation did not change, but as I was changing the number of nodes, it drastically changed the results.

Analysis

Which algorithm performed better?

The algorithm that performed best was the decision tree. I'm sure if I put more time in finding the optimal number of nodes in the hidden layers I could find a much better neural network, but for now, we will settle for a decision tree

Compare accuracy, recall, and precision metrics by class.

The algorithm with the best accuracy, recall, and precision is the decision tree. The algorithm with the second best was the logistic regression. Both algorithms did have very high values (being >.87), so both could technically be used for predictions well

Why do you think the better-performing algorithm did better?

I think the decision tree did better because of the amount of variables that we are using for prediction.

R or Sklearn (be blunt).

Being completely honest, I think I prefer sklearn over R. I can point very strongly to the fact that using sklearn can be accessed on the cloud. I had some down time over the weekend but didn't have my laptop on me, and it was broken that I could access this homework on a different computer. Also the interface of Google CoLab is really streamlined and very clearly a Google product. I was worried when switching over to Python ML that it would be harder to execute than R, but it seemed relatively the same. It was great to get experience with both and do not think learning R was a waste of time.