Untitled

November 6, 2022

[1]:
```
import pandas
import seaborn
```

[2]:
```
df = pandas.read_csv('Auto.csv')
print('First 5 rows:')
print(df.iloc[:5])
print('Dimensions:')
print(df.shape)
```

```
First 5 rows:
    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:
(392, 9)
```

[3]:
```
print('Describe MPG:')
print(df['mpg'].describe())
print('Describe weight:')
print(df['weight'].describe())
print('Describe year:')
print(df['year'].describe())
```

```
Describe MPG:
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
Describe weight:
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
Describe year:
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
```

MPG Range: 37.6, average: 23.445918 Weight range: 3527, average: 2977.584184 Year range: 12, average: 76.010256

[4]:
```
print(df.dtypes)
```

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

[5]:
```
df['cylinders'] = df['cylinders'].astype('category').cat.codes
```

[6]:
```
df['origin'] = df['origin'].astype('category')
```

[7]:
```
print('Cylinders dtype: ' + str(df['cylinders'].dtype))
print('Origin dtype: ' + str(df['origin'].dtype))
```

```
Cylinders dtype: int8
```

```
Origin dtype: category
```

[8]:
```python
df = df.dropna()
print('New dimensions:')
print(df.shape)
```

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

[9]:
```python
avg_mpg = df['mpg'].describe()['mean']
df['mpg_high'] = df['mpg'].apply(lambda x: 1 if x > avg_mpg else 0).
 ↪astype('category')
```

[10]:
```python
if 'mpg' in df:
    del df['mpg']
if 'name' in df:
    del df['name']
```

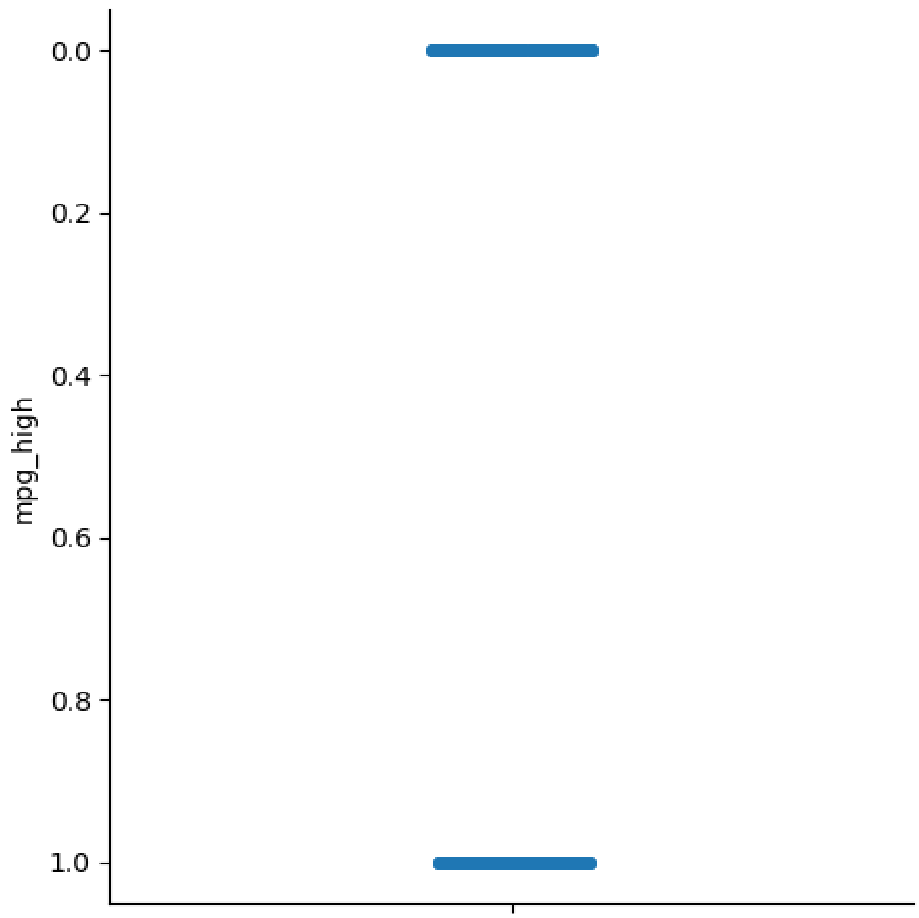
[11]:
```python
print('First 5 rows:')
print(df.iloc[:5])
```

```
First 5 rows:
   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
```

[12]:
```python
seaborn.catplot(df['mpg_high'])
```

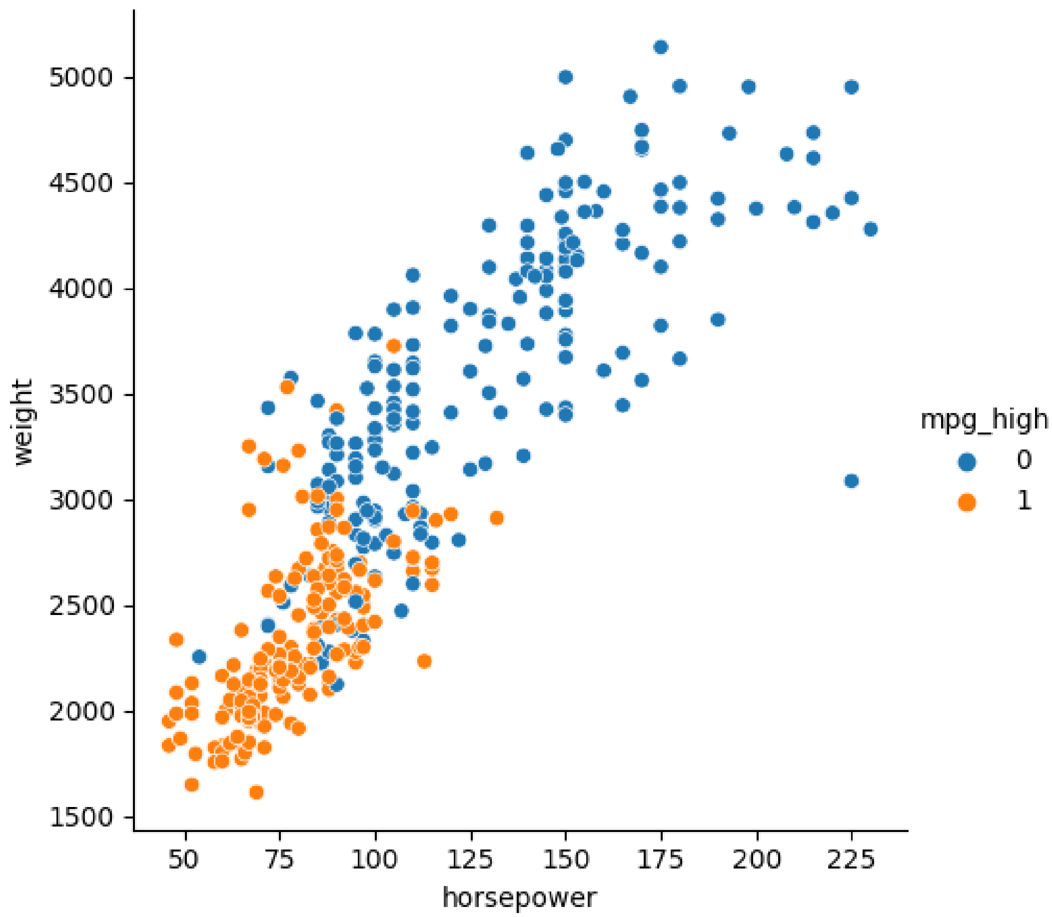
[12]:
```
<seaborn.axisgrid.FacetGrid at 0x1eee13395e0>
```



Comment: mpg_high has both zero and one values and nothing in between (obviously, since we defined these categories ourselves)

[13]:
```
seaborn.relplot(df, x='horsepower', y='weight', hue='mpg_high')
```

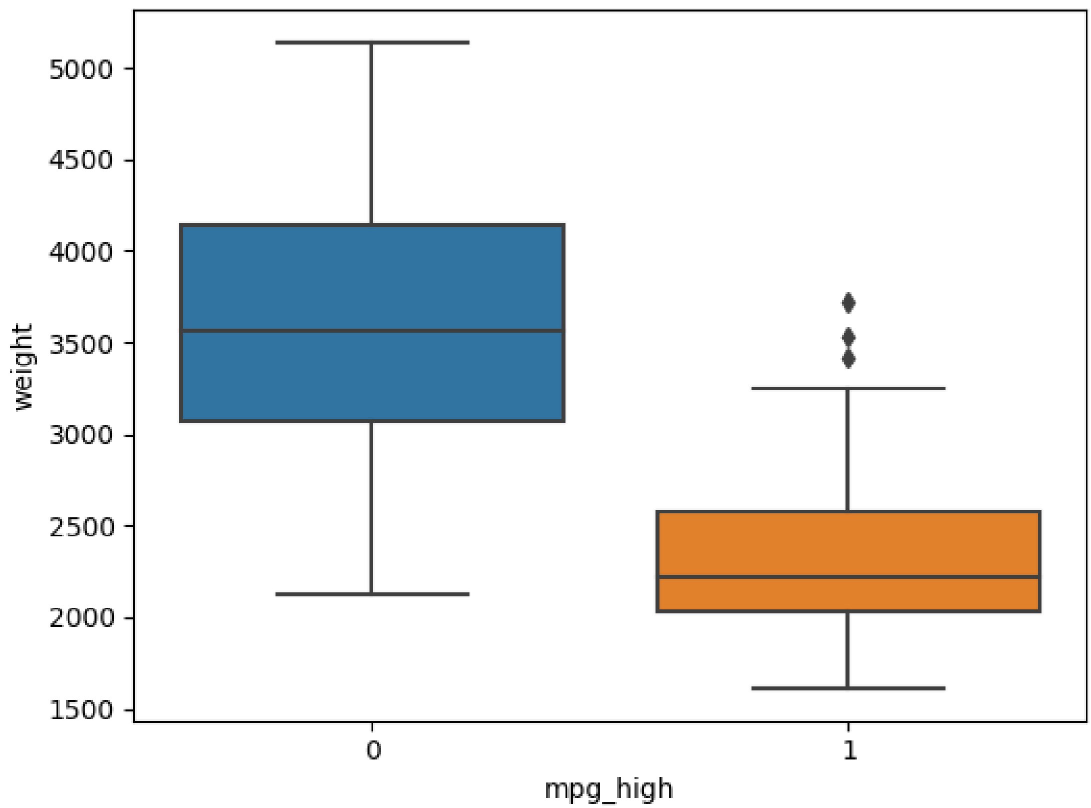
[13]: <seaborn.axisgrid.FacetGrid at 0x1eee12c78b0>



Comment: High MPG vehicles seem to be more clustered towards the low end of the weight/horsepower spectrum (as expected), while there is a much greater variety of low MPG vehicles.

[14]:
```
seaborn.boxplot(df, x='mpg_high', y='weight')
```

[14]: <AxesSubplot: xlabel='mpg_high', ylabel='weight'>



Comment: The box plot confirms the difference in variety between low and high MPG vehicles that was noted above.

[15]:
```
import sklearn
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(df, df['mpg_high'],␣
↪test_size=0.2, random_state=1234)
del X_train['mpg_high']
print('Train dimensions:')
print(X_train.shape)
del X_test['mpg_high']
print('Test dimensions:')
print(X_test.shape)
```

```
Train dimensions:
(311, 7)
Test dimensions:
(78, 7)
```

[16]:
```
from sklearn import preprocessing
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
```

```python
from sklearn.pipeline import make_pipeline
from sklearn.metrics import classification_report
pipe = make_pipeline(StandardScaler(), LogisticRegression(solver='lbfgs')) #␣
 ↪Without scaling, lbfgs doesn't converse
pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.82      0.90        50
           1       0.76      1.00      0.86        28

    accuracy                           0.88        78
   macro avg       0.88      0.91      0.88        78
weighted avg       0.91      0.88      0.89        78
```

[17]:
```python
from sklearn import tree
pipe = make_pipeline(StandardScaler(), tree.
 ↪DecisionTreeClassifier(random_state=1234))
pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)
print(classification_report(y_test, y_pred))
```

```
              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
```

[18]:
```python
from sklearn.neural_network import MLPClassifier

pipe = make_pipeline(StandardScaler(), MLPClassifier(solver='lbfgs',␣
 ↪hidden_layer_sizes=(7, 3), random_state=1234))
pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)
print('Network 1:')
print(classification_report(y_test, y_pred))

pipe = make_pipeline(StandardScaler(), MLPClassifier(solver='lbfgs',␣
 ↪hidden_layer_sizes=(9, 15), random_state=1234))
pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)
```

```python
print('Network 2:')
print(classification_report(y_test, y_pred))
```

```
Network 1:
              precision    recall  f1-score   support

           0       0.92      0.92      0.92        50
           1       0.86      0.86      0.86        28

    accuracy                           0.90        78
   macro avg       0.89      0.89      0.89        78
weighted avg       0.90      0.90      0.90        78

Network 2:
              precision    recall  f1-score   support

           0       0.98      0.92      0.95        50
           1       0.87      0.96      0.92        28

    accuracy                           0.94        78
   macro avg       0.92      0.94      0.93        78
weighted avg       0.94      0.94      0.94        78
```

Performance was somewhat dependent on the values of hidden_layer_sizes but seemed to hover in the range 0.88-0.92 in many cases. Increasing the number of hidden layers would intuitively make the model better, but after a certain point it begins to overfit and performance drops. (9, 15) were the best settings I found in a reasonable amount of time.

0.1 Analysis

The (9, 15) neural network algorithm performed best, while the decision tree algorithm was almost identical in performance. For all of teh accuracy, precision, and recall metrics, the (9, 15) neural network performs best, followed by the decision tree, followed by the (7, 3) neural network, and worst of all is the logistic regression algorithm. The best neural network most likely outperformed the others as neural networks are extremely powerful and can learn all sorts of functions; nevertheless, "in real life" the decision tree might be better for such a small dataset, as neural networks are prone to overfitting and are less efficient to train (though that was not noticeable here).

Python and Scikit-Learn are much more enjoyable to use than R, as Python feels more like a "real" programming language and follows many conventions similar to other languages. At the same time, Python and libraries such as Pandas add a huge amount of "syntactic sugar" that allows you to write very expressive, shorthand code if you know what you're doing.