Untitled

April 8, 2023

[1]:
```
import pandas as pd

data = pd.read_csv('Auto.csv')
print(data.shape)
data.head()
```

```
(392, 9)
```

[1]:

	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

[2]:
```
data[['mpg','weight','year']].describe()
```

[2]:

	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 average of each column is 23.44, 2977.58, and 76, the range of each is 37, 3527, and 12; both respectively for mpg, weight, and year.

[3]:
```
print(f"before:\n{data.dtypes}\n")
data.cylinders = data.cylinders.astype('category').cat.codes
```

```
data.origin = data.origin.astype('category')
print(f"after:\n{data.dtypes}")
```

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

after:
mpg              float64
cylinders           int8
displacement     float64
horsepower         int64
weight             int64
acceleration     float64
year             float64
origin          category
name              object
dtype: object
```

[4]:
```
del_rows = [i for i, x in enumerate(data.isnull().itertuples()) if True in x]
data = data.drop(del_rows)
print(data.shape)
```

```
(388, 9)
```

[5]:
```
import numpy as np

avg_mpg = np.mean(data.mpg)
data['mpg_high'] = [1 if x > avg_mpg else 0 for x in data.mpg]
data.mpg_high = data.mpg_high.astype('category')
data = data.drop(columns=['mpg','name'])
data.head()
```

[5]:

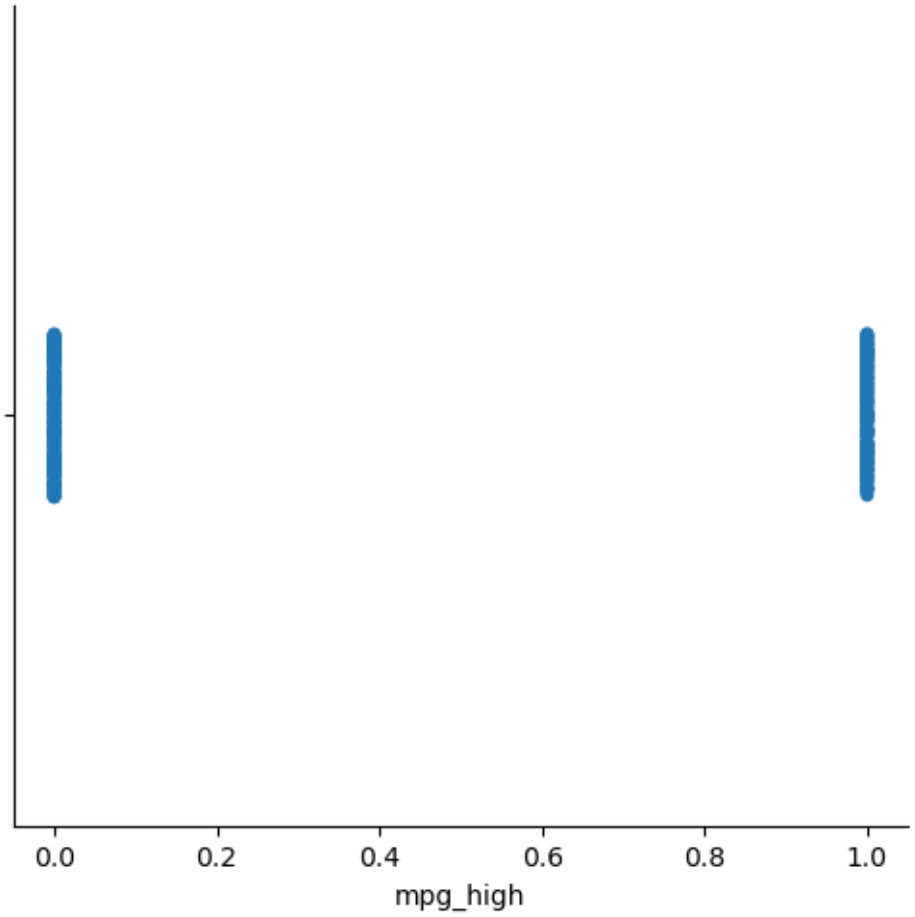
	cylinders	displacement	horsepower	weight	acceleration	year	origin	\
0	4	307.0	130	3504	12.0	70.0	1	
2	4	318.0	150	3436	11.0	70.0	1	
3	4	304.0	150	3433	12.0	70.0	1	
6	4	454.0	220	4354	9.0	70.0	1	
7	4	440.0	215	4312	8.5	70.0	1	

```
   mpg_high
0         0
2         0
3         0
6         0
7         0
```

[6]:
```
%matplotlib inline
import seaborn as sb

# There is not much to be gained here. Just two variables that are indicating␣
 ↪mpg being higher or lower on average.
sb.catplot(x='mpg_high', data=data)
```

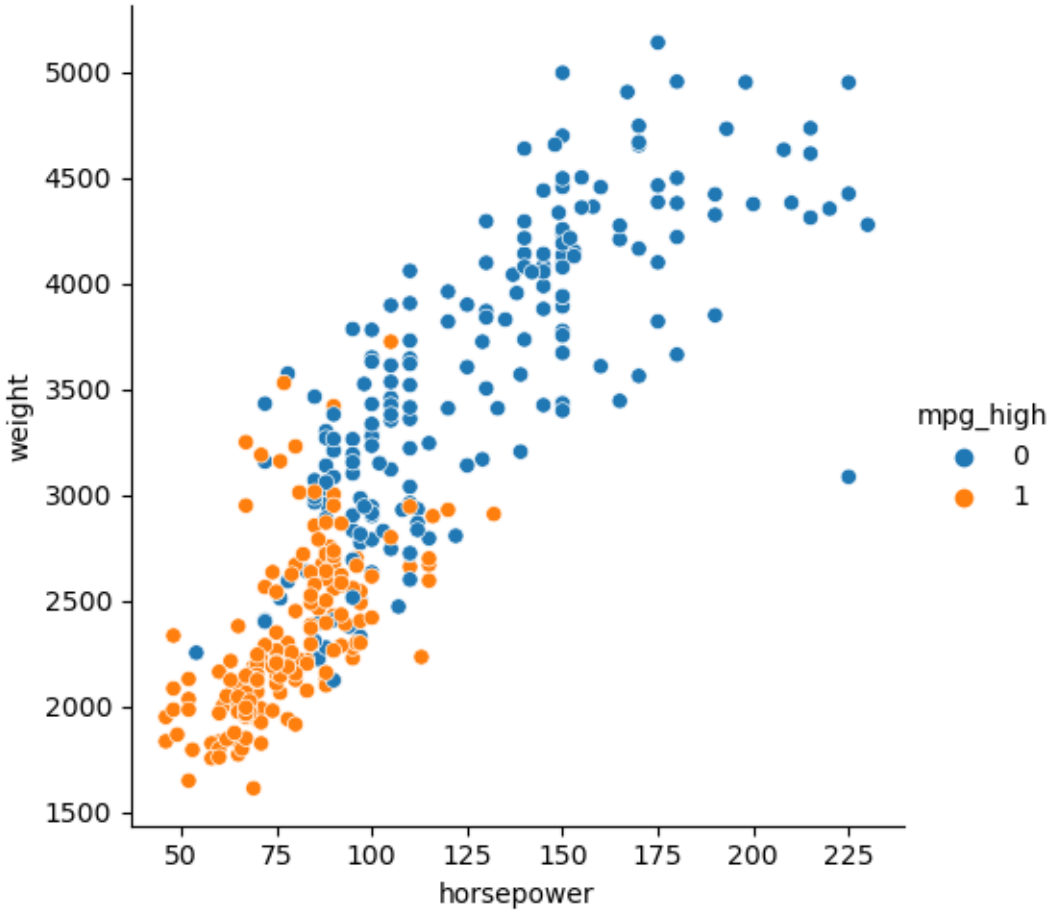
[6]:
```
<seaborn.axisgrid.FacetGrid at 0x7f369b2f68d0>
```



[7]:
```
# This shows that, on average, the lower the weight and horsepower, the more␣
 ↪likely that a car will have a higher than average MPG.
```

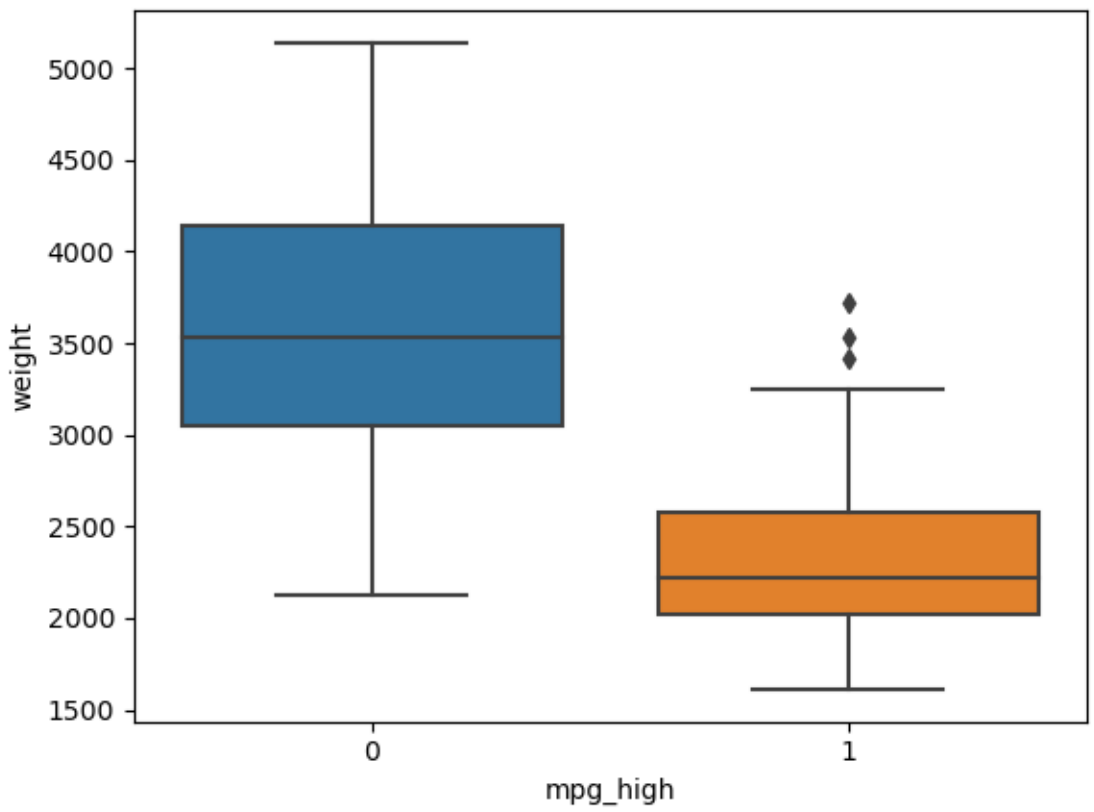
```
sb.relplot(x='horsepower',y='weight',hue='mpg_high', data=data)
```

[7]: <seaborn.axisgrid.FacetGrid at 0x7f369b106210>



[8]:
```
# THis shows that the weight typically will be significantly lower than average␣
↪if the car is MPG efficient.
sb.boxplot(x='mpg_high',y='weight',data=data)
```

[8]: <Axes: xlabel='mpg_high', ylabel='weight'>



[9]:
```
from sklearn.model_selection import train_test_split as tts
from sklearn.metrics import classification_report as report

X, X_t, y, y_t = tts(data[['cylinders',
'displacement',
'horsepower',
'weight',
'acceleration',
'year',
'origin']], np.ravel(data[['mpg_high']]), test_size=0.2, random_state=1234)
print(f'X: {X.shape}\nX_t: {X_t.shape}\ny: {y.shape}\ny_t: {y_t.shape}')
```

```
X: (310, 7)
X_t: (78, 7)
y: (310,)
y_t: (78,)
```

[10]:
```
# logreg
from sklearn.linear_model import LogisticRegression as LR
lr = LR(solver='lbfgs', random_state=1)
lr.fit(X, y)
```

```
preds = lr.predict(X_t)
print(report(y_t, preds))
```

```
              precision    recall  f1-score   support

           0       0.97      0.79      0.87        47
           1       0.75      0.97      0.85        31

    accuracy                           0.86        78
   macro avg       0.86      0.88      0.86        78
weighted avg       0.88      0.86      0.86        78


/usr/lib64/python3.11/site-packages/sklearn/linear_model/_logistic.py:444:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[11]:
```
# DecisionTree
from sklearn.tree import DecisionTreeClassifier as DTC
dtc = DTC(random_state=1)
dtc.fit(X, y)
preds = dtc.predict(X_t)
print(report(y_t, preds))
```

```
              precision    recall  f1-score   support

           0       0.95      0.87      0.91        47
           1       0.83      0.94      0.88        31

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

[12]:
```
# NN
from sklearn.neural_network import MLPClassifier as MLPC
from sklearn.preprocessing import StandardScaler as SS

scale = SS().fit(X)
X_s, X_ts = scale.transform(X), scale.transform(X_t)

mlpc_0 = MLPC(
```

```
    solver='lbfgs',
    max_iter=1000,
    random_state=1,
    hidden_layer_sizes=(16,),
    activation='relu',
    early_stopping=True,
)
mlpc_0.fit(X_s, y)
preds = mlpc_0.predict(X_ts)
print(report(y_t, preds))
```

```
              precision    recall  f1-score   support

           0       0.98      0.89      0.93        47
           1       0.86      0.97      0.91        31

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

[13]:
```
mlpc_1 = MLPC(
    solver='lbfgs',
    max_iter=1000,
    random_state=1,
    hidden_layer_sizes=(32,4,),
    activation='logistic',
    early_stopping=True,
)
mlpc_1.fit(X_s, y)
preds = mlpc_1.predict(X_ts)
print(report(y_t, preds))
```

```
              precision    recall  f1-score   support

           0       0.96      0.94      0.95        47
           1       0.91      0.94      0.92        31

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

Both of the models had equal creation and testing performance. The first model had less overall accuracy than the second model, but both of them were very close to each other. The main difference in the models was the activation and not the model topology, strangely enough. This was confirmed by changing the activation in the second model to 'relu', and seeing all of the scores in the report decrease by about 5%. The logistic activation was probably better for this problem because it fixes all the values between 0 and 1, which is typical for classification functions of this

caliber.

The second neural network seemed to perform the best out of all 4 of the models, as it's accuracy is the highest of all the models. While the first neural network's "low mpg" precision was nearly perfect, it's recall suffered, meaning it probably overfitted on the lower metric in comparision. This is similar to the logistic regression and the decision tree. As for the second neural network, it seems to have actually found some good, advanced correlations within the data.

As for working with R vs `sklearn` (python), I didn't like R in the first place, so I am horribly biased. SKlearn isn't as terrible to work with compared to how "magic" R is. R prides itself on it's magic for statisticians, and as a (iirc this is what it's called) "domain specific language" it fit's the bill quite well. In comparision, python is significantly less "magic," and so is `sklearn`.

As for what I mean by "magic," I'll define it as "the language does something expected in an unexpected way, or in a way significantly different from other languages." Something like how passing by value/object in python works, or slicing using `c()` in R.