```
### Francisco Trejo

### ML with sklearn


### Read Auto Data

import pandas as pd
import numpy as np
import seaborn as sb

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)
```

```
### Data Exploration

df[["mpg","weight","year"]].describe(include="all")

# MPG Average = 23.45 Range = (9 - 46.6)
# Weight Average = 2977.58 Range = (1613 - 5140)
# Year Average = 76.01 Range = (70 - 82)
```

	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

+ Code + Text

```
### Explore Data Types

df.dtypes
```

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

```
df.cylinders = df.cylinders.astype("category").cat.codes
df.origin = df.origin.astype("category")
df.dtypes
```

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

```
### Deal with NA's

df = df.dropna()
print('\nDimensions of dataframe:', df.shape)
```

```
Dimensions of dataframe: (389, 9)
```

```
### Modify Columns

average = df['mpg'].mean()
df['mpg_high'] = np.where(df['mpg'] > average , 1, 0)
df.mpg_high = df.mpg_high.astype('category')
df = df.drop(columns=['mpg','name'])
print(df)
```

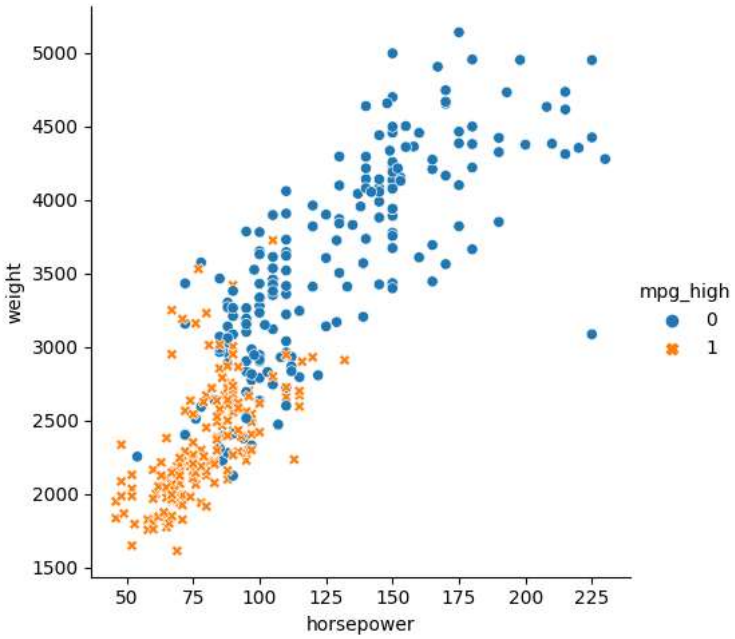
```
     cylinders  displacement  horsepower  weight  acceleration  year origin  \
0            4         307.0         130    3504          12.0  70.0      1
1            4         350.0         165    3693          11.5  70.0      1
2            4         318.0         150    3436          11.0  70.0      1
3            4         304.0         150    3433          12.0  70.0      1
6            4         454.0         220    4354           9.0  70.0      1
..         ...           ...         ...     ...           ...   ...    ...
387          1         140.0          86    2790          15.6  82.0      1
388          1          97.0          52    2130          24.6  82.0      2
389          1         135.0          84    2295          11.6  82.0      1
390          1         120.0          79    2625          18.6  82.0      1
391          1         119.0          82    2720          19.4  82.0      1

    mpg_high
0          0
1          0
2          0
3          0
6          0
..       ...
387        1
388        1
389        1
390        1
391        1

[389 rows x 8 columns]
```

```
### Data Exploration with Graphs

sb.catplot(x="mpg_high", kind='count', data=df)

# One thing we learn from this graph is that there are less vehichles with a mpg over the average compared to ones that are lower to the aver
```

```
<seaborn.axisgrid.FacetGrid at 0x7fe3fa44fcd0>
```

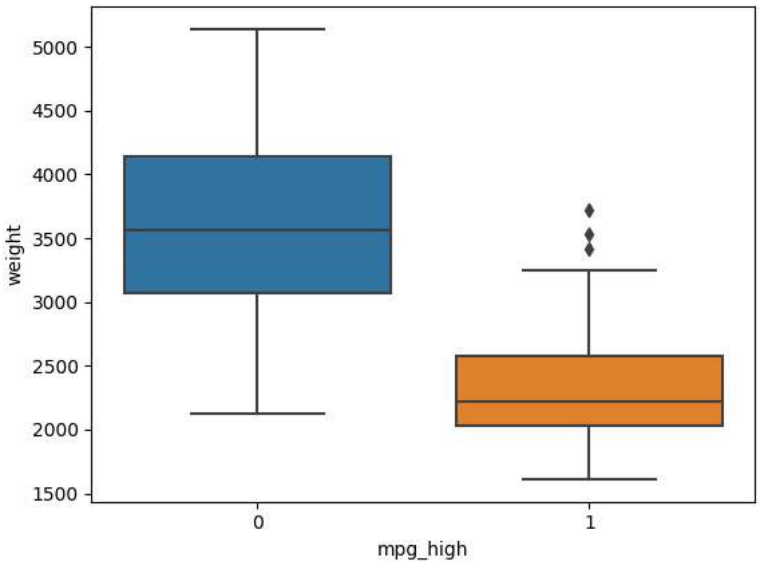

```
sb.relplot(x='horsepower', y='weight', data=df, hue=df.mpg_high, style=df.mpg_high)

# We learn in the graph that cars that have higher mpg compared to the average are more condensed in the area that has lower horserpower and
```

```
<seaborn.axisgrid.FacetGrid at 0x7fe3fa6d54c0>
```




```
sb.boxplot(x='mpg_high', y='weight', data=df)

# We learn from this graph that cars with lower mpg than the average has more weight distribution compared to cars that have mpg higher than
```

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




```
### Train/Test Split
```

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

```
train size: (311, 7)
test size: (78, 7)
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
import time
st=time.time()
clf = LogisticRegression(solver='lbfgs', max_iter=1000)
clf.fit(X_train, Y_train)
pred = clf.predict(X_test)
et = time.time()
t = et - st
clf.score(X_train, Y_train)
```

```
0.9035369774919614
```

```
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))
print('algo time: ', t)
```

```
accuracy score:  0.8974358974358975
precision score:  0.7777777777777778
recall score:  1.0
f1 score:  0.8750000000000001
algo time:  0.029597759246826172
```

```
from sklearn.metrics import confusion_matrix

confusion_matrix(Y_test, pred)
```

```
array([[42,  8],
       [ 0, 28]])
```

```
from sklearn.metrics import classification_report
print(classification_report(Y_test, pred))
```

```
                  precision    recall  f1-score   support

               0       1.00      0.84      0.91        50
               1       0.78      1.00      0.88        28

        accuracy                           0.90        78
       macro avg       0.89      0.92      0.89        78
    weighted avg       0.92      0.90      0.90        78
```

Decesion Tree

```
from sklearn.tree import DecisionTreeClassifier
st = time.time()
clf = DecisionTreeClassifier()
clf.fit(X_train, Y_train)
pred = clf.predict(X_test)
et = time.time()
t = et - st

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))
print('algo time: ', t)
```

```
print(classification_report(Y_test, pred))
```

```
     accuracy score:  0.9230769230769231
     precision score:  0.84375
     recall score:  0.9642857142857143
     f1 score:  0.8999999999999999
     algo time:  0.007536649703979492
                  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
```

```
confusion_matrix(Y_test, pred)
```

```
     array([[45,  5],
            [ 1, 27]])
```

```
### Neural Networks

from sklearn import preprocessing
# Normalizing the data
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

st = time.time()
clf = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5, 2), max_iter=500, random_state=1234)
clf.fit(X_train_scaled, Y_train)

pred = clf.predict(X_test_scaled)
et = time.time()
t = et - st

print('accuracy = ', accuracy_score(Y_test, pred))
print('algo time: ', t)

confusion_matrix(Y_test, pred)
```

```
     accuracy =  0.9102564102564102
     algo time:  0.19280791282653809
     array([[46,  4],
            [ 3, 25]])
```

```
print(classification_report(Y_test, pred))
```

```
                  precision    recall  f1-score   support

               0       0.94      0.92      0.93        50
               1       0.86      0.89      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
```

```
# Different topolgy and settings

clf = MLPClassifier(solver='adam', hidden_layer_sizes=(10, 7), max_iter=1000, random_state=1234)
clf.fit(X_train_scaled, Y_train)

pred = clf.predict(X_test_scaled)

print('accuracy = ', accuracy_score(Y_test, pred))
```

```
confusion_matrix(Y_test, pred)
```

```
accuracy =  0.8717948717948718
array([[43,  7],
       [ 3, 25]])
```

```
print(classification_report(Y_test, pred))

# For the first settings we got .91 accuracy and for the second we got .87 accuracy
# We changed all the 3 settings. The solver, layer size, and iterations, so all had some type of effect on the accuracy
# I think it was mostly the layer size that had an impact. Since we used more nodes and layers it tends to overfit the data which has an impa
```

```
                   precision    recall  f1-score   support

               0       0.93      0.86      0.90        50
               1       0.78      0.89      0.83        28

        accuracy                           0.87        78
       macro avg       0.86      0.88      0.86        78
    weighted avg       0.88      0.87      0.87        78
```

Analysis

The algorithm that worked the best was the one with more accuracy which was both Neural Networks and Decision Trees at .91. Logistic wasn't too far off and and had an accuracy of .90. However, in terms of speed Neural Networks took about .19 seconds which was the slowest out of the three. Logestic Regression took .03 and Decesions Trees turned out to be the fasted with .01. With a high accuracy and a low speed, Decesion Trees turned out to be the best algorithm. The reason why this could of been better is because of the small data set and we had a small data set. Some algorithms perform better or worse depending on the size of the data set. Plus we know from before that Neural Networks performace can also be atrributed to it's parameters like layer size, type of solver, and number of iterations. We can get a higer accuracy if we manipulate this, but run time will still be slow. It is a trade off one has to think about when using these algorithms. My experience with SK learn felt the same as R. Both have useful libraries that can be used with machine learning algorithms. My only negative is the way to download the pakages you need to do the algorithms. For SK learn you don't have to really download them beforehand. All you have to do is just write code to import them when you need them just like R. However, R you have to download the pakages in order to import them. Maybe thats just because of we used R Studios versus Google Colab but it can be annoying sometimes.