sklean in Python

Read in auto data

```
import pandas as pd
import sklearn
df = pd.read_csv("https://raw.githubusercontent.com/RyanBanafshay/Machine_Learning_Portfolio/main/ML%20with%20sklearn/Auto.csv")
df.head()
```

	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

Output the dimensions

```
df_ndim = df.shape
print("Number of dimensions: ", df_ndim)
```

```
Number of dimensions:  (392, 9)
```

Data Exploration

```
print("MPG:\n\n",df['mpg'].describe())
print("\nWeight:\n\n",df['weight'].describe())
print("\nYear:\n\n",df['year'].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

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

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
```

Year has two less fields than MPG and Weight. MPG and Weight also have a relative small standard deviation. Year has a smaller range.

```
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 = df.astype({"origin":'category'})
df.dtypes
```

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

Drop NA values from the data

```
df=df.dropna()
df.isna().sum()
```

```
mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    0
year            0
origin          0
name            0
dtype: int64
```

Add Column

```
import numpy as np
avg = df.mpg.mean()
df['mpg_high'] = np.where(df.mpg > avg, 1, 0)
```

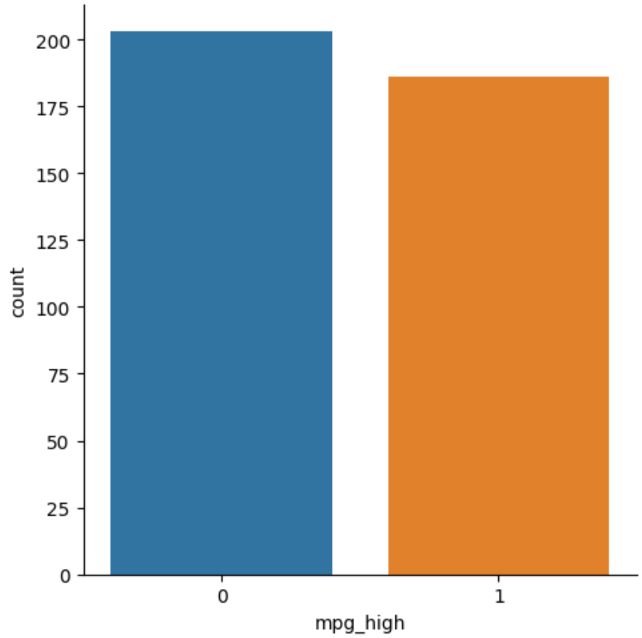
```
df = df.drop(columns=['name','mpg'])
print(df.head())
```

```
   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
```

Graphing the data

```
import seaborn as sb

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

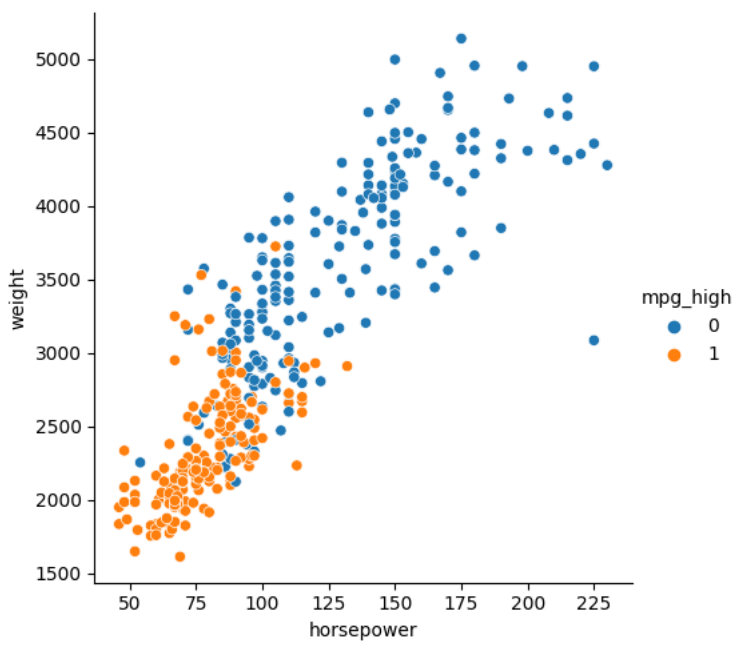
```
<seaborn.axisgrid.FacetGrid at 0x7f7bfa5c6d90>
```



As we see from the visual representation, there seems to be very little difference in the data for mpg_high.

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

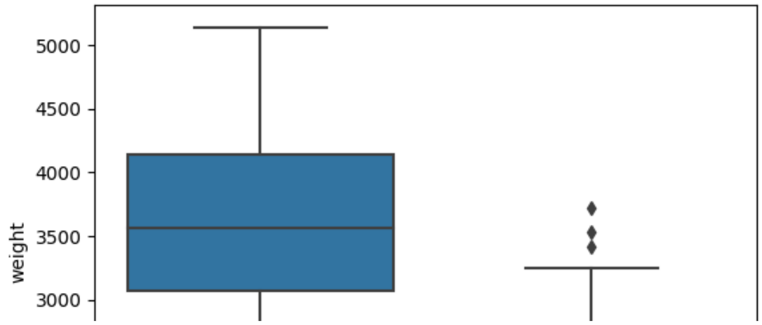
```
<seaborn.axisgrid.FacetGrid at 0x7f7bfa48f3d0>
```



Horsepower and Weight much higher foor the 0 mpg_high. We also see that the weight typically corresponds with a higher horsepower.

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

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



This boxplot shows us very similar results as the previous. There are a few outliers on mpg_high 1

Training the data

```
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=0.3, random_state=21, stratify=y)
print("Training size: ", X_train.shape)
print("Testing size: ", X_test.shape)
```

```
Training size:  (272, 6)
Testing size:  (117, 6)
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression

logReg = LogisticRegression(solver = 'lbfgs', max_iter=100000)
logReg.fit(X_train, y_train)
logReg.score(X_train, y_train)
```

```
0.9007352941176471
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier

dTree = DecisionTreeClassifier()
dTree.fit(X_train, y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
predDT = dTree.predict(X_test)

accuracyDT = accuracy_score(y_test, predDT)
precisionDT = precision_score(y_test, predDT)
recallDT = recall_score(y_test, predDT)
f1DT = f1_score(y_test, predDT)


print("accuracy score: ", accuracyDT)
print("precision score: ", precisionDT)
print("recall score: ", recallDT)
print("f1 score: ", f1DT)
```

```
accuracy score:  0.9316239316239316
precision score:  0.9137931034482759
recall score:  0.9464285714285714
f1 score:  0.9298245614035087
```

Neural Network

```
from sklearn import preprocessing

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

nn1 = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5, 2), max_iter=500, random_state=1234)
nn1.fit(X_train_scaled, y_train)
```

```
▾                              MLPClassifier
MLPClassifier(hidden_layer_sizes=(5, 2), max_iter=500, random_state=1234,
              solver='lbfgs')
```

```
prednn1 = nn1.predict(X_test_scaled)
```

```
from sklearn.metrics import confusion_matrix

accuracynn1 = accuracy_score(y_test, prednn1)
precisionnn1 = precision_score(y_test, prednn1)
recallnn1 = recall_score(y_test, prednn1)
f1nn1 = f1_score(y_test, prednn1)

print("accuracy score: ", accuracynn1)
print("precision score: ", precisionnn1)
print("recall score: ", recallnn1)
print("f1 score: ", f1nn1)

confusion_matrix(y_test, prednn1)
```

```
accuracy score:  0.9230769230769231
precision score:  0.9122807017543859
recall score:  0.9285714285714286
f1 score:  0.9203539823008849
array([[56,  5],
       [ 4, 52]])
```

Second Neural Network

```
nn2 = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(4, 2), max_iter=500, random_state=1234)
nn2.fit(X_train_scaled, y_train)
```

```
▾                              MLPClassifier
MLPClassifier(hidden_layer_sizes=(4, 2), max_iter=500, random_state=1234,
              solver='lbfgs')
```

```
prednn2 = nn2.predict(X_test_scaled)
```

```
accuracynn2 = accuracy_score(y_test, prednn2)
precisionnn2 = precision_score(y_test, prednn2)
recallnn2 = recall_score(y_test, prednn2)
f1nn2 = f1_score(y_test, prednn2)

print("accuracy score: ", accuracynn2)
print("precision score: ", precisionnn2)
print("recall score: ", recallnn2)
print("f1 score: ", f1nn2)

confusion_matrix(y_test, prednn2)
```

```
accuracy score:  0.9145299145299145
precision score:  0.9107142857142857
recall score:  0.9107142857142857
f1 score:  0.9107142857142857
```

```
array([[56,  5],
       [ 5, 51]])
```

In the second model, I lowered the number of hidden layers. This was ineffective at increasing the accuracy. Overall performance of the second model was generally worse than the first. I am assuming that the difference in both these models are minimal and not really reflective of how effective they are because the data size is so small.

Analysis

The decision tree had the most accurate results, though not by such a signiciant amount that I would say its the best algorithm. They all performed well with the data. A real test to see which would be better would be to use a variety of different and larger data sets and compare them using that.

If we compare the different metrics by class, we see:

Decision Tree:

- Accuracy = 0.9316239316239316
- Precision = 0.9137931034482759
- Recall = 0.9464285714285714

NN1:

- Accuracy = 0.9230769230769231
- Precision score = 0.9122807017543859
- Recall score = 0.9285714285714286

NN2:

- Accuracy = 0.9145299145299145
- Precision = 0.9107142857142857
- Recall = 0.9107142857142857

What algorithm to use is dependent on the data. I think a big reason why decision tree preformed the best for this particular data is because decision trees are effective at handling non-linear datasets. Logisitc regression had the worst accuracy, and in that case it needs a linear dataset to be fully effective. The nueral network performed well but I think the main reason why it didn't prefer as well is because NN's are more suited for larger more complex datasets.

Honestly, I prefer using sklearn in python over R. Python was designed to be easy to use for programmers used to more convential programming languages such as Java or C++. Although R was convinent in how it was able to formulate results through internal functions, I never got used to the conventions of the language. Simple things like assigning variables felt more complicated than it needed to be. At a higher level, I would say both sklearn and R are very similar with no real preference one way or another. So the real defining reason why I prefer sklearn is simply that I find python a more familiar platform that has many similarity to Java and C++ with added conveniences.