Huy Nguyen

HNN190000

CS 4375.004

Portfolio Assignment: ML with sklearn

1. Read the Auto data (5 points)

a. use pandas to read the data

```
import numpy as np
import pandas as pd
import seaborn as sb
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn import tree
from sklearn import preprocessing
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_scor
from sklearn.metrics import classification_report
import io

from google.colab import files
uploaded = files.upload()

df = pd.read_csv(io.StringIO(uploaded['Auto.csv'].decode('utf-8')))
```

```
Browse... Auto.csv
Auto.csv(application/vnd.ms-excel) - 17859 bytes, last modified: n/a - 100% done
Saving Auto.csv to Auto (5).csv
```

b. output the first few rows

```
df.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	r
0	18.0	8	307.0	130	3504	12.0	70.0	1	chev che ma

1	15.0	8	350.0	165	3693	11.5	70.0	1	b sky
2	18.0	8	318.0	150	3436	11.0	70.0	1	plym sat

c. output the dimensions of the data

```
print('\nDimensions of data frame:', df.shape)
```

```
     Dimensions of data frame: (392, 9)
```

2. Data exploration with code (5 points)

a. use describe() on the mpg, weight, and year columns

[ ] ↳ *2 cells hidden*

3. Explore data types (5 points)

a. check the data types of all columns

```
df.dtypes
```

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

b. change the cylinders column to categorical (use cat.codes)

```
df.cylinders = df.cylinders.astype('category').cat.codes
```

c. change the origin column to categorical (don't use cat.codes)

```
df.origin = df.origin.astype('category')
df.origin = df.origin.astype('category')
```

d. verify the changes with the dtypes attribute

```
df.dtypes
```

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

4. Deal with NAs (5 points

a. delete rows with NAs

```
df = df.dropna()
```

b. output the new dimensions

```
print('\nDimensions of data frame:', df.shape)
```

```
     Dimensions of data frame: (389, 9)
```

5. Modify columns (10 points)

a. make a new column, mpg_high, and make it categorical:

i. the column == 1 if mpg > average mpg, else == 0

```
df.insert(0, 'mpg_high', 0)
df.mpg_high = df.mpg_high.astype('category').cat.codes
df = df.reset_index()
for x in range(389):
  if df.loc[x, 'mpg'] > df['mpg'].mean():
    df.loc[x, 'mpg_high'] = 1
  else:
    df.loc[x, 'mpg_high'] = 0
```

```
<ipython-input-170-dfaf1f69fca7>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
  df.mpg_high = df.mpg_high.astype('category').cat.codes
```

b. delete the mpg and name columns (delete mpg so the algorithm doesn't just learn to predict mpg_high from mpg)

```
df = df[['mpg_high', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration',
```

c. output the first few rows of the modified data frame

```
df
```

	mpg_high	cylinders	displacement	horsepower	weight	acceleration	year	origi
0	0	4	307.0	130	3504	12.0	70.0	
1	0	4	350.0	165	3693	11.5	70.0	
2	0	4	318.0	150	3436	11.0	70.0	
3	0	4	304.0	150	3433	12.0	70.0	
4	0	4	454.0	220	4354	9.0	70.0	
...	...	...	...	...	...	...	...	..
384	1	1	140.0	86	2790	15.6	82.0	
385	1	1	97.0	52	2130	24.6	82.0	:
386	1	1	135.0	84	2295	11.6	82.0	
387	1	1	120.0	79	2625	18.6	82.0	
388	1	1	119.0	82	2720	19.4	82.0	

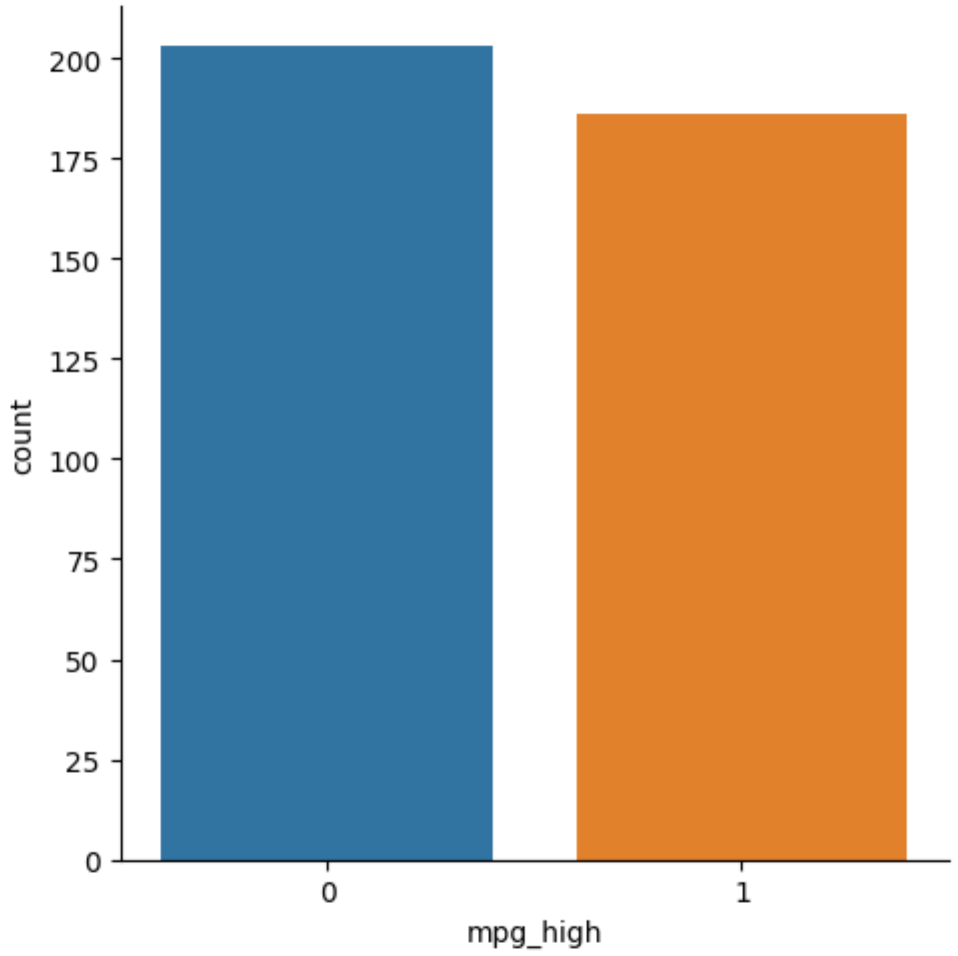
389 rows × 8 columns

6. Data exploration with graphs (15 points)

a. seaborn catplot on the mpg_high column

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

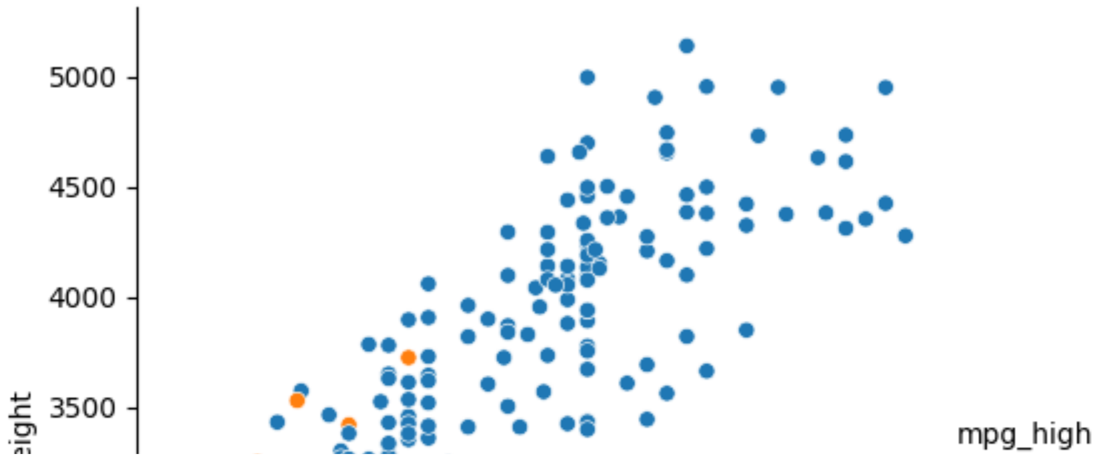
```
<seaborn.axisgrid.FacetGrid at 0x7f185cf4fee0>
```

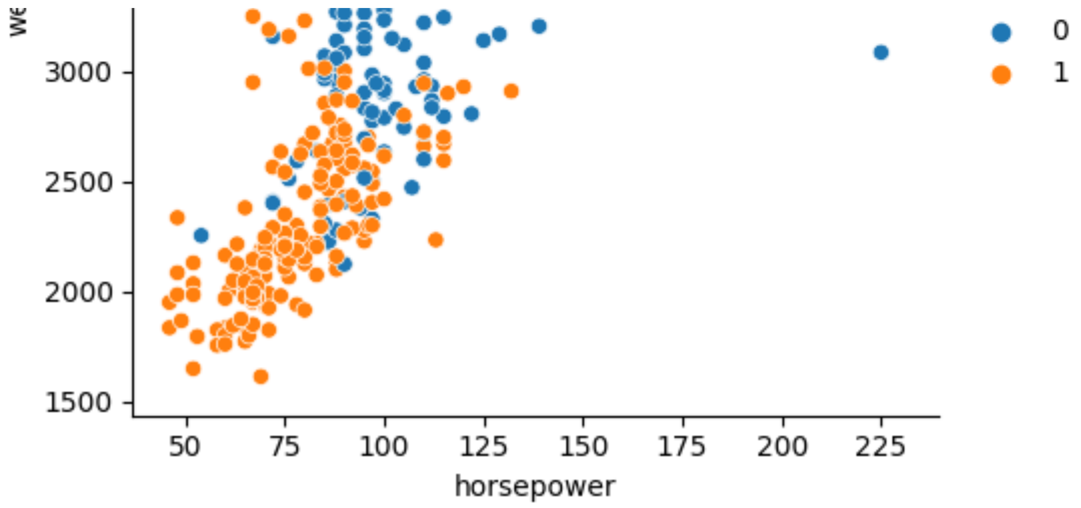


b. seaborn relplot with horsepower on the x axis, weight on the y axis, setting hue or style to mpg_high

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

```
<seaborn.axisgrid.FacetGrid at 0x7f185cb11040>
```

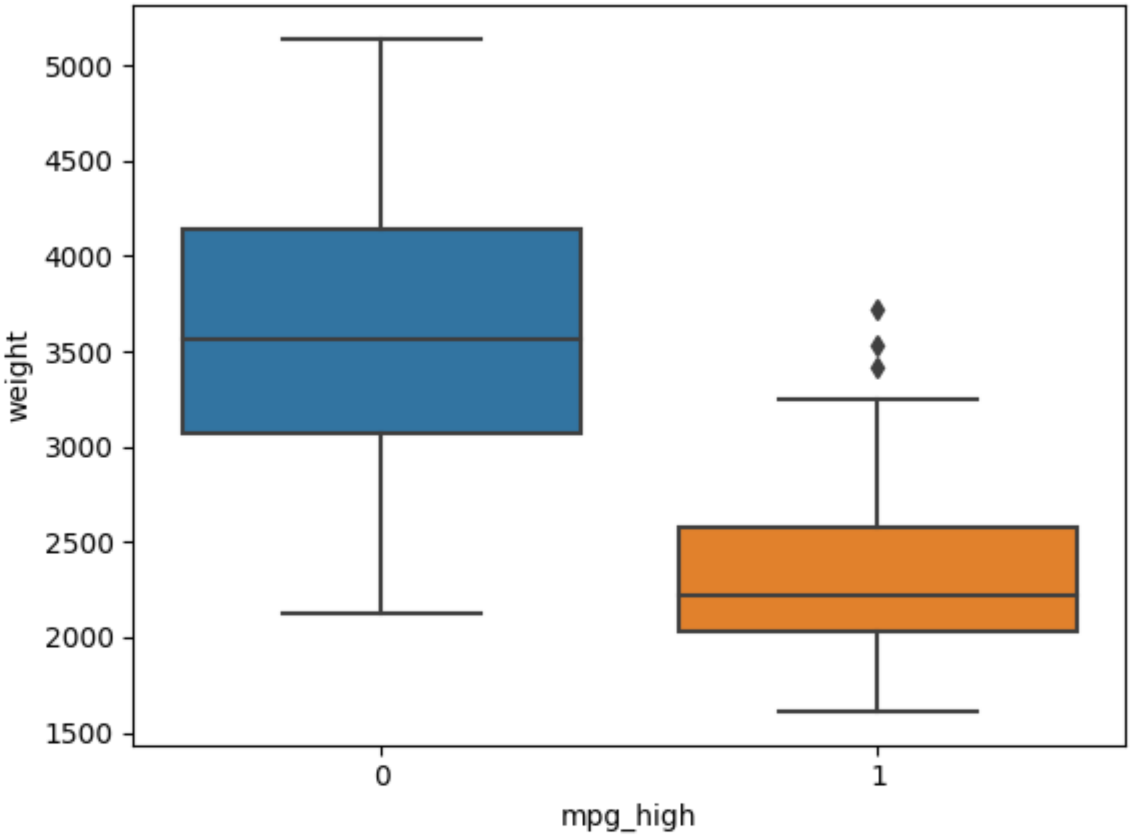




c. seaborn boxplot with mpg_high on the x axis and weight on the y axis

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

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



d. for each graph, write a comment indicating one thing you learned about the data from the graph

The bar graph showed us how many row were above and below the average mpg. I noticed that

The bar graph showed us how many row were above and below the average mpg. I noticed that there were a little more rows that had less than the average.

The scatterplot showed us that smaller cars with less horsepower had a mpg greater than the mean of mpg.

The boxplot showed us any outliers when mpg was greater than the mean. It also shows how much variance is in the data. There is less variance in the mpg that is greater than the mean compared to less than the mean.

7. Train/test split (5 points)

a. 80/20

```
X_train, X_test, y_train, y_test = train_test_split(df.loc[:, ['cylinders', 'displacement',
```

b. use seed 1234 so we all get the same results

```
I set the random_state to 1234 to have consistent results
```

c. train /test X data frames consists of all remaining columns except mpg_high

```
I set all the columns except for mpg_high as the X and mph_high as the Y
```

d. output the dimensions of train and test

```
print('\nDimensions of X_train:', X_train.shape)
print('\nDimensions of X_test:', X_test.shape)
print('\nDimensions of y_train:', y_train.shape)
print('\nDimensions of y_test:', y_test.shape)
```

```
    Dimensions of X_train: (311, 7)

    Dimensions of X_test: (78, 7)

    Dimensions of y_train: (311,)
```

```
Dimensions of y_test: (78,)
```

8. Logistic Regression (10 points)

a. train a logistic regression model using solver lbfgs

```
clf = LogisticRegression()
clf.fit(X_train, y_train)
clf.score(X_train, y_train)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: Converg
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(
0.9067524115755627
```

b. test and evaluate

```
pred = clf.predict(X_test)

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))
```

```
accuracy score:  0.8589743589743589
precision score:  0.7297297297297297
recall score:  0.9642857142857143
f1 score:  0.8307692307692307
```

c. print metrics using the 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
```

9. Decision Tree (10 points)

a. train a decision tree

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

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

b. test and evaluate

```
pred = clf.predict(X_test)

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))
```

```
accuracy score:  0.9358974358974359
precision score:  0.8709677419354839
recall score:  0.9642857142857143
f1 score:  0.9152542372881356
```

c. print the classification report metrics

```
print(classification_report(y_test, pred))
```

```
              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
```

d. plot the tree (optional, see: https://scikit-learn.org/stable/modules/tree.html)

10. Neural Network (15 points)

a. train a neural network, choosing a network topology of your choice

```
#scale data
scaler = preprocessing.StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

#training
clf = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(6,), max_iter=1500, random_state=1
clf.fit(X_train_scaled, y_train)
```

```
▾                              MLPClassifier
MLPClassifier(hidden_layer_sizes=(6,), max_iter=1500, random_state=1234,
              solver='lbfgs')
```

b. test and evaluate

```
pred = clf.predict(X_test_scaled)
print('accuracy = ', accuracy_score(y_test, pred))

confusion_matrix(y_test, pred)
print(classification_report(y_test, pred))
```

```
accuracy =  0.8974358974358975
              precision    recall  f1-score   support

           0       0.94      0.90      0.92        50
           1       0.83      0.89      0.86        28

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

c. train a second network with a different topology and different settings

```
#changed changed hidden layer sizes to (6,2) from (6,)
clf = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(6,2), max_iter=1500, random_state=
clf.fit(X_train_scaled, y_train)
```

```
▾                              MLPClassifier
```

```
MLPClassifier(hidden_layer_sizes=(6, 2), max_iter=1500, random_state=1234,
              solver='lbfgs')
```

d. test and evaluate

```
pred = clf.predict(X_test_scaled)
print('accuracy = ', accuracy_score(y_test, pred))

confusion_matrix(y_test, pred)
print(classification_report(y_test, pred))
```

```
accuracy =  0.8589743589743589
              precision    recall  f1-score   support

           0       0.95      0.82      0.88        50
           1       0.74      0.93      0.83        28

    accuracy                           0.86        78
   macro avg       0.85      0.87      0.85        78
weighted avg       0.88      0.86      0.86        78
```

e. compare the two models and why you think the performance was same/different

The model first model had 1 hidden layer with 6 hidden units while the second model had 2 hidden layers with 6 hidden unit. I noticed that the accuracy decreased a little after gaining another hidden layer. This may be because this data set is small and the extra hidden layer may have made it more inaccurate.

11. Analysis (15 points)

a. which algorithm performed better?

```
The decision tree model gave me the highest accuracy out of all the models.
```

b. compare accuracy, recall and precision metrics by class

Summarizing the results from each model

Logistic Regression

```
accuracy score:  0.8589743589743589

precision score:  0.7297297297297297

recall score:  0.9642857142857143

f1 score:  0.8307692307692307
```

Decision Tree

```
accuracy score:  0.9230769230769231

precision score:  0.8666666666666667

recall score:  0.9285714285714286

f1 score:  0.896551724137931
```

Neural Network

1 Hidden Layer, 6 Hidden Units

```
accuracy =  0.8974358974358975
          precision    recall  f1-score   support


       0       0.94      0.90      0.92        50
       1       0.83      0.89      0.86        28


accuracy                           0.90
```

2 Hidden Layer, 6 Hidden Units

```
accuracy =  0.8589743589743589
          precision    recall  f1-score   support
       0       0.95      0.82      0.88        50
       1       0.74      0.93      0.83        28
accuracy                           0.86        78
```

Overall decision tree gave me the highest accuracy with a 0.92. Neural Network did give me the highest precision of 0.95 and 0.94 when labeling 0. Logistic Classification gave me the highest

highest precision of 0.95 and 0.94 when labeling 0. Logistic Classification gave me the highest recall with 0.96.

c. give your analysis of why the better-performing algorithm might have outperformed the other

Even though Decision trees had the highest accuracy, the other models were relatively close to that accuracy as well. Decision Trees may have overperformed because I could have incorrectly modeled the other models which gave it the slightly higher accuracy.

d. write a couple of sentences comparing your experiences using R versus sklearn. Feel free to express strong preferences.

I preferred using sklearn compared to R. I think this because I liked using google colab compared to Rstudios ide. I also find it nice that I do not have to download R and Rstudio when I can just use Google colab on a browser. I also was more accustomed to python compared to R.

There also was very funny setting that I liked on google colab that allowed little cats and dogs run around top of the page.