Haniyyah Hamid | ML with skLearn

Reading in Data

```
# importing libraries
import pandas as pd
import numpy as np
import seaborn as sb
from sklearn import datasets
from google.colab import drive
drive.mount('/content/gdrive')

# reading in data
df = pd.read_csv('gdrive/My Drive/Auto.csv')

# printing first few rows and dimensions of the df
print(df.head())
print('\nDimensions of data frame:', df.shape)
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly
remount, call drive.mount("/content/gdrive", force_remount=True).
    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()
```

```
              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
```

Exploring Data Types

```
df.dtypes
```

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

```
# changing data type with cat.codes

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

print(df.dtypes, "\n")
print(df.head())
```

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

    mpg  cylinders  displacement  horsepower  weight  acceleration
year  \
0  18.0          4         307.0         130    3504          12.0
70.0
1  15.0          4         350.0         165    3693          11.5
70.0
2  18.0          4         318.0         150    3436          11.0
```

```
70.0
3  16.0          4         304.0         150    3433           12.0
70.0
4  17.0          4         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
```

```
# changing data type without cat.codes
df.origin = df.origin.astype('category')

print(df.dtypes, "\n")
print(df.head())
```

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

    mpg  cylinders  displacement  horsepower  weight  acceleration
year  \
0  18.0          4         307.0         130    3504           12.0
70.0
1  15.0          4         350.0         165    3693           11.5
70.0
2  18.0          4         318.0         150    3436           11.0
70.0
3  16.0          4         304.0         150    3433           12.0
70.0
4  17.0          4         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
```

Removing NAs

```
# checking number of NAs in each column

df.isnull().sum()
```

```
mpg             0
cylinders       0
displacement    0
horsepower      0
weight          0
acceleration    1
year            2
origin          0
name            0
dtype: int64
```

```
# dropping the rows with NAs

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

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

Modifying Columns

```
# create new column
df['mpg_high'] = np.where(df['mpg'] > df['mpg'].mean(), 1, 0)

# make column a 'category' type
df.mpg_high = df.mpg_high.astype('category').cat.codes

# delete rows
del df["mpg"]
del df["name"]

# outputting first few rows of new df
print(df.dtypes, "\n")
print(df.head())
```

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

```
   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

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py: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/user_guide/indexing.html#
returning-a-view-versus-a-copy

/usr/local/lib/python3.7/dist-packages/pandas/core/generic.py:5516:
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/user_guide/indexing.html#
returning-a-view-versus-a-copy
  self[name] = value
```

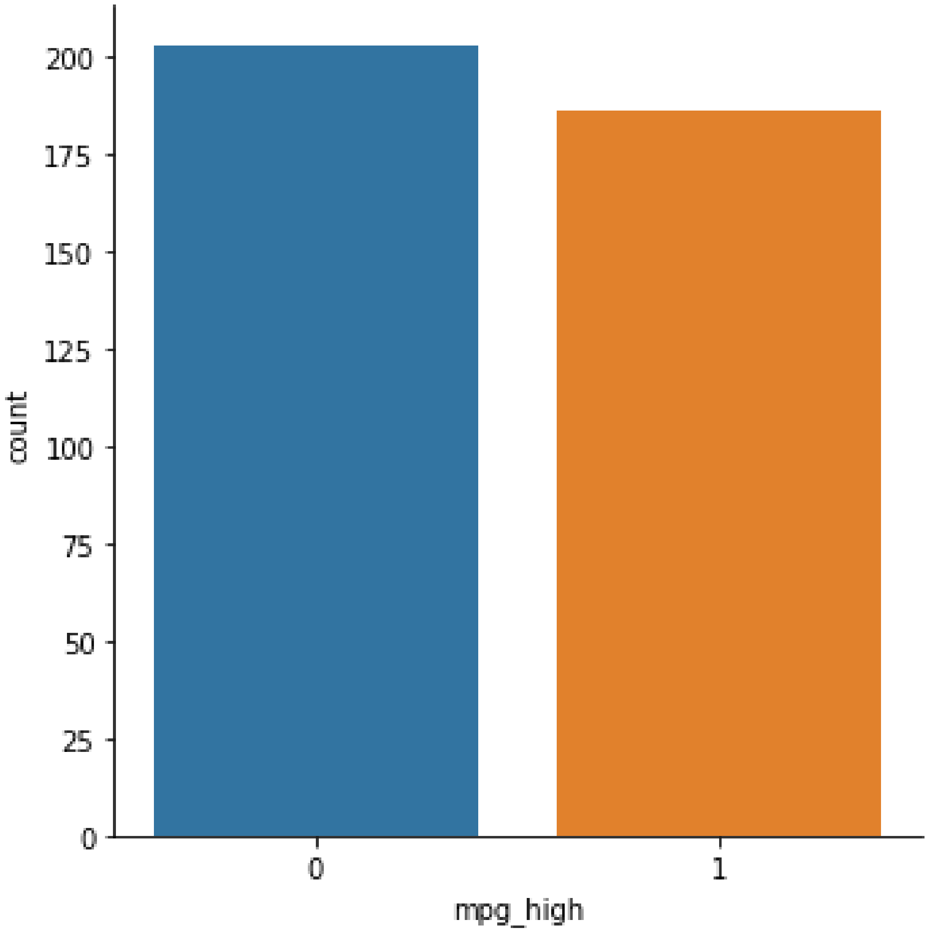
Data Exploration with Graphs

```
import seaborn as sb
from sklearn import datasets

# cat plot of mpg_high column

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

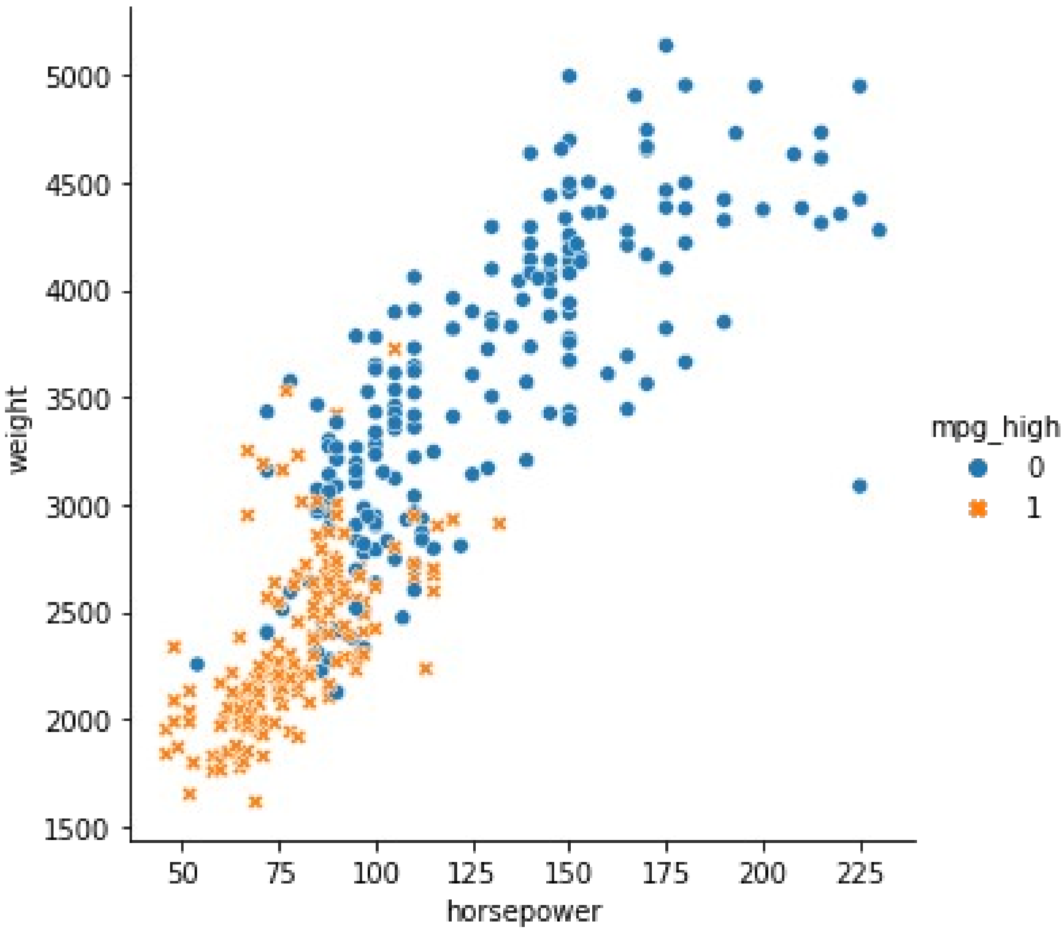
```
<seaborn.axisgrid.FacetGrid at 0x7f9522323bd0>
```



We can observe that there's more cars that have a mpg that is less than the average mpg.

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

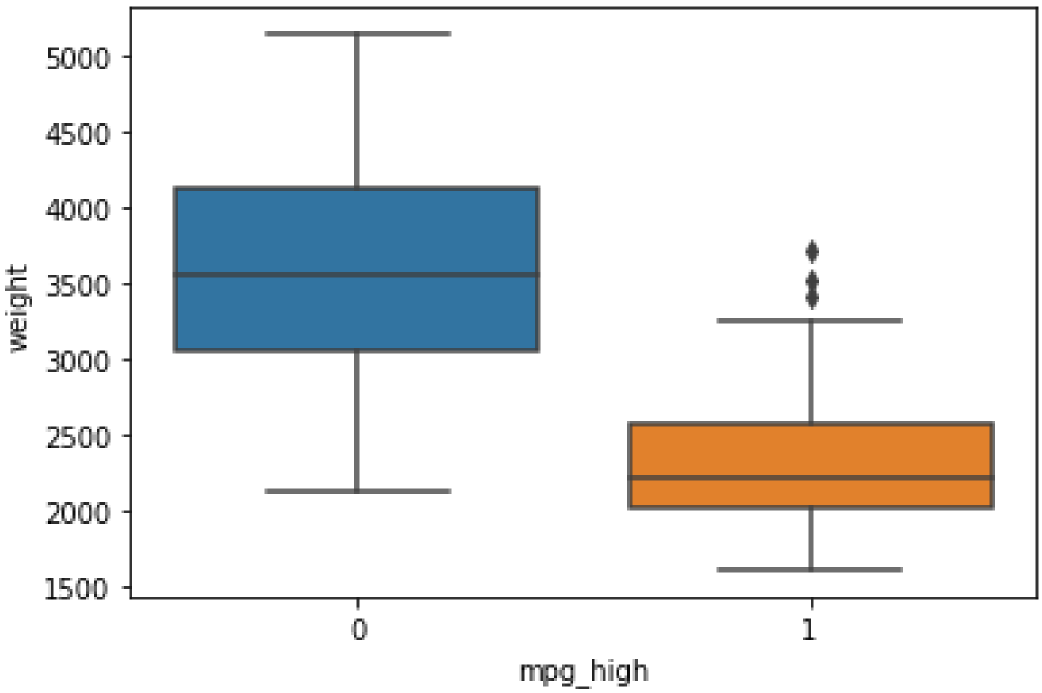
```
<seaborn.axisgrid.FacetGrid at 0x7f9521c04450>
```



We can observe that the higher the horsepower, the more weight a car has.

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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9521b66590>
```



We can observe that the median weight of cars who have an mpg less than the average mpg is higher than ones that are greater than the average mpg.

Training Data

```
# 80/20 train test split
from sklearn.model_selection import train_test_split

# with seed 1234
X = df.loc[:, ['cylinders', 'displacement', 'horsepower', 'weight',
'acceleration', 'year', 'origin']]
y = df.mpg_high
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=1234)

# outputting dimensions of training and test dataframes
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

# training
clf = LogisticRegression()
```

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

# testing/prediction
pred = clf.predict(X_test)

# evaluation
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, confusion_matrix, classification_report

print('Accuracy: ', accuracy_score(y_test, pred))
print('Precision: ', precision_score(y_test, pred))
print('Recall: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))

# classification report
print(classification_report(y_test, pred, target_names=None))

# confusion matrix
confusion_matrix(y_test, pred)
```

```
Accuracy:  0.8589743589743589
Precision:  0.7297297297297297
Recall:  0.9642857142857143
f1 score:  0.8307692307692307
              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


/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/
_logistic.py:818: 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
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

```
array([[40, 10],
       [ 1, 27]])
```

Decision Tree

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

# training
clf2 = DecisionTreeClassifier()
clf2.fit(X_train, y_train)
clf2.score(X_train, y_train)

# testing/prediction
y_pred = clf2.predict(X_test)

print('Accuracy: ', accuracy_score(y_test, y_pred))
print('Precision: ', precision_score(y_test, y_pred))
print('Recall: ', recall_score(y_test, y_pred))
print('f1 score: ', f1_score(y_test, y_pred))

# classification report
print(classification_report(y_test, y_pred, target_names=None))

# confusion matrix
confusion_matrix(y_test, y_pred)

# plot tree (optional)
#tree.plot_tree(clf2)
```

```
Accuracy:  0.9102564102564102
Precision:  0.8387096774193549
Recall:  0.9285714285714286
f1 score:  0.8813559322033899
              precision    recall  f1-score   support

           0       0.96      0.90      0.93        50
           1       0.84      0.93      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


array([[45,  5],
       [ 2, 26]])
```

Neural Network

```
from sklearn import preprocessing

# normalizing data w/ preprocessing functions

scaler = preprocessing.StandardScaler().fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)

# training data for neural network
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(solver='lbfgs', hidden_layer_sizes=(5, 2),
max_iter=500, random_state=1234)
clf.fit(X_train_scaled, y_train)

# testing/prediction
pred = clf.predict(X_test_scaled)

# evaluation
print('Accuracy: ', accuracy_score(y_test, pred))
print('Precision: ', precision_score(y_test, pred))
print('Recall: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))

# classification report
print(classification_report(y_test, pred))

# confusion matrix
confusion_matrix(y_test, pred)
```

```
Accuracy:  0.8717948717948718
Precision:  0.78125
Recall:  0.8928571428571429
f1 score:  0.8333333333333334
              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


array([[43,  7],
       [ 3, 25]])
```

```python
# training w/ different topology settings
clf = MLPClassifier(solver='sgd', hidden_layer_sizes=(3,),
max_iter=1500, random_state=1234)
clf.fit(X_train_scaled, y_train)

# testing/prediction
pred = clf.predict(X_test_scaled)

# evaluation
print('Accuracy: ', accuracy_score(y_test, pred))
print('Precision: ', precision_score(y_test, pred))
print('Recall: ', recall_score(y_test, pred))
print('f1 score: ', f1_score(y_test, pred))

# classification report
print(classification_report(y_test, pred))

# confusion matrix
confusion_matrix(y_test, pred)
```

```
Accuracy:  0.8333333333333334
Precision:  0.7142857142857143
Recall:  0.8928571428571429
f1 score:  0.7936507936507937
              precision    recall  f1-score   support

           0       0.93      0.80      0.86        50
           1       0.71      0.89      0.79        28

    accuracy                           0.83        78
   macro avg       0.82      0.85      0.83        78
weighted avg       0.85      0.83      0.84        78


array([[40, 10],
       [ 3, 25]])
```

After changing the topology settings, we see that the second neural network was less accurate and less precise. Which would mean that the first neural network was a better suited algorithm for this data.

Analysis

The logistic regression and decision tree algorithms seem to have performed better than the neural networks. For class 0: logistic regression gave the best precision, decision tree gave the best recall score. For class 1: decision tree gave the best precision, logistic regression gave the best recall score. The decision tree and first neural network gave the best accuracies. Overall, I think logistic regression and decision tree algorithms outperformed the neural networks. This may most likely be due to the fact that neural networks can be more likely to overfit data than those two algorithms. Even after modifying the settings of the first neural network, the algorithm still didn't perform better than the other two. I prefer sklearn over R mostly because of the simplicity of implementation. In terms of training, evaluating training data, gathering the metrics report (in sklearn its classification_report) and plotting, I thought it exceeded R.